COMPUTER-AIDED DECISION MAKING FOR OCEAN TOWING

by

TODD JAY PELTZER

S.B. Naval Architecture and Marine Engineering
Massachusetts Institute of Technology
(1979)

SUBMITTED TO THE DEPARTMENT OF OCEAN ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREES OF

NAVAL ENGINEER

and

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June, 1989

© Todd Jay Peltzer, 1989. All rights reserved

The author hereby grants to M.I.T. and to the U.S. Government permission to reproduce and to distribute copies of this thesis document in whole or in part.

T246000

Thesis
P3445
C.1

THESIS
P5746

COMPUTER AIDED DECISION MAKING

FOR OCEAN TOWING

by

TODD JAY PELTZER

Submitted to the Department of Ocean Engineering
on May 12, 1989 in partial fulfillment of the
requirements for the degrees of Naval Engineer and
Master of Science in Mechanical Engineering

ABSTRACT

An interactive program for microcomputers, TOWCALC, was developed to implement the results of recent theoretical work in nonlinear towing dynamics. This work makes possible a statistical description of extreme tensions in towlines based on the seakeeping motions of the tug and the tow, where the extreme tension is defined as the sum of static tension and peak dynamic tension which has a probability of 0.001 of occurring in a given day of towing. The resulting data base of extreme tensions (for a range of tugs, tows, towing speeds, towline lengths, sea states, and wave angles) is incorporated into TOWCALC, which automates the process of estimating mean towline tension and extreme towline tension. Mean towline tension is estimated using methods given in the U.S. Navy Towing manual for resistance of the tow, and an analytic method is developed for the towline resistance. TOWCALC gives tow planners the ability to quickly evaluate the feasibility of a given tow, as well as to evaluate the level of risk; tug operators at sea will be able to anticipate dangerous peak tensions and take steps to reduce the risk of towline failure. As a result, using TOWCALC will lead to safer and more efficient ocean towing.

Thesis Supervisor: Jerome H. Milgram
Title: Professor of Ocean Engineering

Acknowledgements

I would like to thank the Department of the Navy for making my graduate studies possible, the Office of Naval Research for supporting this research, and the office of the U.S. Navy Supervisor of Salvage and Diving for their technical support and encouragement.

I would also like to express my appreciation to the people who helped make this work a success:

To Professor Jerome Milgram, my thesis supervisor, my sincerest appreciation for your enthusiasm, encouragement, and support.

To Fernando Frimm, for all your patient help, many thanks.

To Norbert Doerry, for your willingness to lend a hand, for your insights into the C language, and for many useful programs ... thanks.

And finally, to my wife Sandy, for your love and support, my deepest gratitude and love; you made this effort worthwhile.

Contents

Chapter 1 Introduction 11

Chapter 2 Towing Dynamics 13

2.1 Introduction 13

2.2 Assumptions 13

2.3 Nonlinear Towline Model 16

2.3.1 Nonlinear Equations of Motion 16

2.3.2 Polynomial Approximation 18

2.4 Towing System Model 19

2.5 Extreme Tension Statistics 24

2.5.1 Short Term Extremal Statistics 25

2.5.2 Computational Method 28

2.5.3 Data Base Development 29

Chapter 3 Resistance Prediction 32

3.1 Tow Resistance 32

3.1.1 Hull resistance 33

3.1.1.1 Self-propelled ships 33

3.1.1.2 Drydocks 33

3.1.1.3 Barges 34

3.1.2 Wave Resistance 36

3.1.2.1 Self-propelled ships 36

3.1.2.2 Drydocks and Barges 39

3.1.3 Wind Resistance 39

3.1.3.1 Self-propelled ships 39

3.1.3.2 Drydocks and Barges 40

3.1.4 Propeller Resistance 40
3.2 Towline Resistance 41
Chapter 4 Program Description 48
4.1 Overview 48
4.1.1 Purpose 48
4.1.2 Intended Users 48
4.1.3 Hardware Requirements 49
4.1.4 Programming Environment 49
4.1.5 User Interface 49
4.1.6 Options 50
4.2 Tug Selection 54
4.3 Tow Selection 62
4.3.1 Entering New Data 62
4.3.1.1 Self-Propelled Ships 65
4.3.1.2 Floating Drydocks 71
4.3.1.3 Barges 76
4.3.2 Editing Existing Data 81
4.3.2.1 Self-Propelled Ships 81
4.3.2.2 Floating Drydocks 82
4.3.2.3 Barges 82
4.3.3 Retrieving Data Files 82
4.3.4 Resistance Predictions 85
4.3.4.1 Self-Propelled Ships 85
4.3.4.2 Floating Drydocks 87
4.3.4.3 Barges 87
4.3.5 Tug Evaluation 87

4.4 Extreme Tension 93
4.4.1 Using the Data Base 93
4.4.2 Computing Extreme Tension 97
4.4.3 Exploring Alternatives 101
4.4.3.1 Effects of Tow Speed 101
4.4.3.2 Effects of Towline Length 103
4.5 Printing a Report 105
Chapter 5 Conclusions 107
5.1 Summary 107
5.2 Further Study 108
Bibliography 109
Appendix A Extreme Tension Data for CVN 65 110
Appendix B Sample Tow Report 114
Appendix C Installing TOWCALC 118
Appendix D TOWCALC Source Code 122

List of Figures

2.1 Towing Bridle Geometry 15

2.2 Tension vs. Towline Span 18

2.3 Towing Geometry 20

2.4 Pierson-Moskowitz Sea Spectrum 20

2.5 Wave Angle Coordinate System 23

2.6 Data Flow for Computational Scheme 29

3.1 Hull Resistance Curves for Self-Propelled Ships 34

3.2 Added Resistance Curves for Self-Propelled Ships 37

3.3 Towing Hawser Geometry 43

3.4 Tangential and Normal Forces on a Cable 43

4.1 TOWCALC Title Screen 52

4.2 Main Program Options 52

4.3 Program Options Menu Structure 53

4.4 Tug Options 55

4.5 Selecting Tug Class 55

4.6 Entering Tug Data 57

4.7 Editing Tug Data 57

4.8 Save Tug File Prompt 59

4.9 Entering Tug File Name 59

4.10 Retrieve Tug Data File 60

4.11 Program Flow: Select Tug Module 61

4.12 Tow Data Options 63

4.13 Select Tow Ship Type 63

4.14 Program Flow: Enter New Data for Ships 64

4.15 Entering Tow Data for Self-Propelled Ships 68
4.16 Propeller Status Menu 68
4.17 Editing Tow Data for Ships 69
4.18 Selecting Ship Class 69
4.19 Editing Ship Data 70
4.20 Select Drydock Class 73
4.21 Entering Hull Condition 73
4.22 Entering Drydock Tow Data 74
4.23 Editing Drydock Tow Data 74
4.24 Program Flow: Enter New Data for Drydocks 75
4.25 Program Flow: Entering New Data for Barges 77
4.26 Entering Barge Dimensions 78
4.27 Barge End Shape Menu 78
4.28 Editing Barge Dimensions 80
4.29 Editing Barge Tow Data 80
4.30 Program Flow: Edit Existing Tow Data 83
4.31 Program Flow: Retrieve Tow Data File 84
4.32 Resistance Summary for Self-Propelled Ships 86
4.33 Resistance Summary for Floating Drydocks 86
4.34 Resistance Summary for Barges 89
4.35 Available Tension vs. Tow Speed 89
4.36 Tug Evaluation Summary and Options Menu 91
4.37 Inadequate Tug Pull Error Message 91
4.38 Confirmation of Speed Selection 92
4.39 Screen Plot of Available Tension vs. Tow Speed 92
4.40 Menu for Selecting Dynamic Tension Basis 96

4.41 Dynamic Tension Summary 96
4.42 Program Flow: Estimate Dynamic Tension 98
4.43 Dynamic Tension Options Menu 99
4.44 Screen Plot of Extreme vs. Mean Towline Tension 99
4.45 Effects of Towing Speed With Options Menu 102
4.46 Screen Plot of Effects of Towing Speed 102
4.47 Effects of Hawser Length With Options Menu 104
4.48 Screen Plot of Effects of Hawser Length 104
4.49 Print Report Options Menu 106

List of Tables

2.1 Data Base Towing Parameters 31

3.1 Drydock Towing Coefficients 35

3.2 Beaufort Scale (From U.S. Navy Towing Manual) 37

3.3 Hawser Resistance (lbs) at 3 knots 47

3.4 Hawser Resistance (lbs) at 6 knots 47

3.5 Hawser Resistance (lbs) at 9 knots 47

4.1 Towing Vessel Characteristics 56

A.1 Tug ARS 38 Towing CVN 65 111

A.2 Tug ARS 50/ATS 1 Towing CVN 65 112

A.3 Tug T-ATF 166 Towing CVN 65 113

Chapter 1

Introduction

As in all maritime endeavors, there is an element of risk in ocean towing. Ship motions due to heavy seas can induce dynamic loads in the towline sufficient to cause failure with possibly disastrous consequences, endangering personnel, hazarding the tug and tow, and, in some cases, endangering the environment.

Managing the risk of towline failure requires some measure of the magnitude of these dynamic loads. Previously, the dynamics of towing with regard to the seakeeping motions of the vessels had not been studied in any detail, so there were no quantitative means to predict towline dynamic tensions. Towing system design used a single factor of safety applied to the mean towline tension to account for dynamic loads; comparison with the towline's nominal breaking strength was then used to evaluate the risk of towline failure.

This simplistic approach has been replaced in Naval practice by the results of recent theoretical work (Frimm [1], Milgram [2]) which make possible a statistical prediction of extreme tension based on the nonlinear towline dynamics. These results predict extreme towline tension (mean plus peak dynamic) based on the mean tension. Using these methods to compute extreme tensions directly is not feasible, however, for tow planners and tug operators. Consequently, a data base of extreme tensions for various combinations of tug, tow, towline length, tow speed, wave angle, and sea state was developed. This has been incorporated into the latest revision of the U.S. Navy Towing Manual [3], consisting of a series of tables and graphs from which a tow planner or a tug operator may determine the extreme tension which has a probability of 0.001 of occurring in a day of towing.

Armed with this knowledge, planners and operators can better evaluate the risks, and thus make more intelligent decisions regarding a given towing situation, leading to safer, more efficient towing. The procedures for computing extreme towline tensions, however, are laborious and time consuming, and thus do not lend themselves to exploring alternatives.

The purpose of this thesis is to provide an interactive, microcomputer based program which automates the process of predicting extreme towline tensions, to be used by U.S. Navy personnel involved in both the planning and execution of ocean tows. This will give tow planners the ability to quickly evaluate the feasibility of a given tow, as well as to evaluate the level of risk. Tug operators at sea will be able to anticipate dangerous peak tensions and take steps to reduce the risk of towline failure.

Chapter 2 discusses the problem of towing dynamics, outlining the theory of nonlinear extreme tensions and their prediction. Chapter 3 discusses the methods used to predict the resistance of the tow and of the towline, which together give the mean towline tension, a pre-requisite for the prediction of extremes. Chapter 4 describes in detail the computer program, **TOWCALC**, developed to implement the results of the recent theoretical developments.

Chapter 2

Towing Dynamics

2.1 Introduction

My purpose here is to outline the problem of towing dynamics, particularly as it relates to the planning and operation of ocean tows. A more thorough and complete treatment of towing dynamics is given by Milgram [2], which forms the theoretical basis for much of this work, and which is summarized here in part.

The goal of the analysis that follows is to determine the magnitude of the dynamic, or time-varying, loads experienced by the towline in order to better predict the possibility of its breaking. To do so, we must form a mathematical model of the complete towing system, which includes the tug, tow, and towline, as well as a description of the environment. The motions of the tug and tow are well described in terms of linear seakeeping theory; the towline tension, however, presents a number of sources of nonlinearity.

In this chapter, I will discuss the nonlinear model for towline tension, the twelve-degree-of-freedom towing system model, the statistics of extreme tensions, and the development of an extreme tension database.

2.2 Assumptions

There are a number of assumptions made in this analysis regarding the composition of the towing system. First, only single tows are considered: multiple tows, although common in practice, are beyond the scope of the present analysis. We consider here only wire rope towlines, as these are the primary towlines used by Navy towing vessels when conducting ocean tows.

The composition of the towline system, which connects the tug and tow, is assumed to be a single shot (90 feet) of chain connected to the towline, with 20 feet of the chain on the deck of the tow; the remaining 70 feet extends past the bow. The chain serves two purposes: it prevents the towline from chafing against the deck edge of the tow, and it reduces the effective stiffness of the catenary. In reality, most Navy tows are rigged with a towing bridle that consists of two shots of chain connected to a "flounder's plate", to which an additional length of chain is attached; this final chain pendant is then connected to the towline. Figure 2.1 shows the two arrangements.

The analysis of nonlinear extreme tensions presented here presumes a fixed towline length. Many Navy towing vessels have constant tension winches, however, which actively control towline length in response to the load. Use of these winches in automatic mode is standard practice, and greatly reduces peak dynamic tensions in the towline. Consequently, the results presented here apply directly to fixed length towlines and will give conservative estimates of dynamic loads when the towing winch is used in automatic mode.

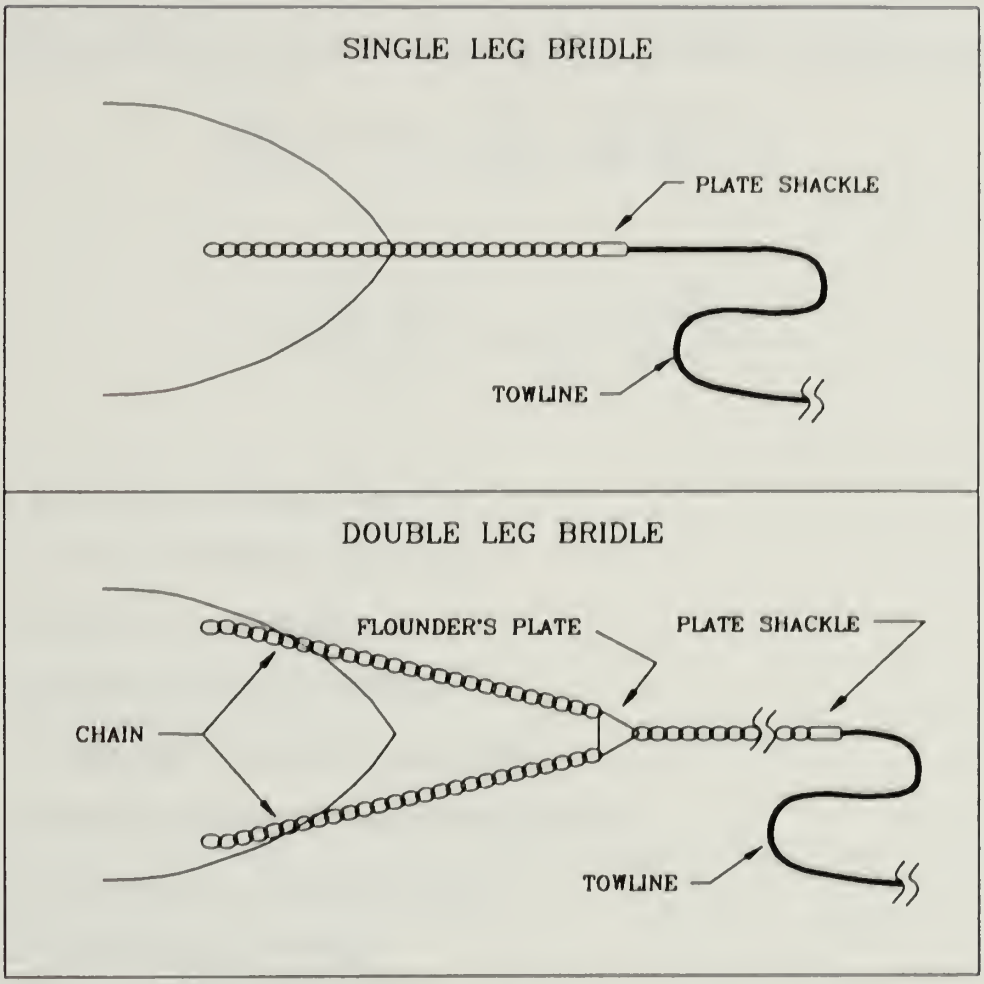

Figure 2.1. Towing Bridle Geometry

2.3 Nonlinear Towline Model

2.3.1 Nonlinear Equations of Motion

The towline is modeled as a cable, that is, as a perfectly flexible curved rod. The governing differential equations for towline dynamics can be written as (Triantafyllou [4])

$$m\frac{\partial^2 q}{\partial t^2} = (\bar{T} + \tilde{T})\left(\alpha + \frac{\partial^2 q}{\partial s^2}\right) - b\frac{\partial q}{\partial t}\left|\frac{\partial q}{\partial t}\right| - \bar{T}\alpha \tag{2.1}$$

with

$$\tilde{T} = EA\left[\frac{p_0}{L} - \frac{\alpha}{L}\int_0^L q\,ds + \frac{1}{2L}\int_0^L \left(\frac{\partial q}{\partial s}\right)^2 ds\right] \tag{2.2}$$

where

A = cross-sectional area of the cable

E = Young's modulus of the towline

m = towline mass per unit length

w = towline weight per unit length

s = Lagrangian coordinate along towline

q, p = normal and tangential motions along towline

$\bar{T}, \tilde{T}$ = static tension and dynamic tension

D, L = cable diameter and length

ρ = mass density of water

p_0 = tangential displacement distribution due to $\bar{T}$

$b = \frac{1}{2}\rho C_D D$ = sectional drag factor

C_D = towline sectional drag coefficient

$\alpha = wL/\bar{T}$ = catenary static curvature

These equations contain some considerable simplifications when compared to a complete dynamical model for cables, but for a towline whose static configuration changes slowly in a seaway the model is adequate. Equations (2.1) and (2.2) preserve the following nonlinearities (Milgram [2]):

1) the crossflow drag force,

2) the product of the dynamic tension and the dynamic curvature,

3) the nonlinear tension-displacement relation for a towline undergoing clipping,

4) the geometric nonlinear tension-extension relationship for a catenary.

The crossflow drag experienced by the towline is due to its transverse motion. At the frequencies of towline end motion excitation found in towing situations, the drag force can dominate inertia forces. This drag opposes the transverse motions, considerably reducing their magnitude, with the net effect that the towline must stretch to accommodate the end motions.

Clipping refers to the phenomenon of a cable that becomes completely slack, and then taut again, as happens during towing. This occurs because the cable cannot support compression.

The dominant nonlinearity in towing dynamics is the relationship between tension and cable extension. Figure 2.2 shows this relationship for a representative towline. Over a relatively large range of cable span, the static tension is a linear function of span. This is because the cable responds to the motion of its end points by changing its geometry—the effect of the catenary. Beyond a certain point, however, the towline can no longer accommodate its end motions in this manner, but must stretch, with the highly nonlinear effects exhibited in Figure 2.2.

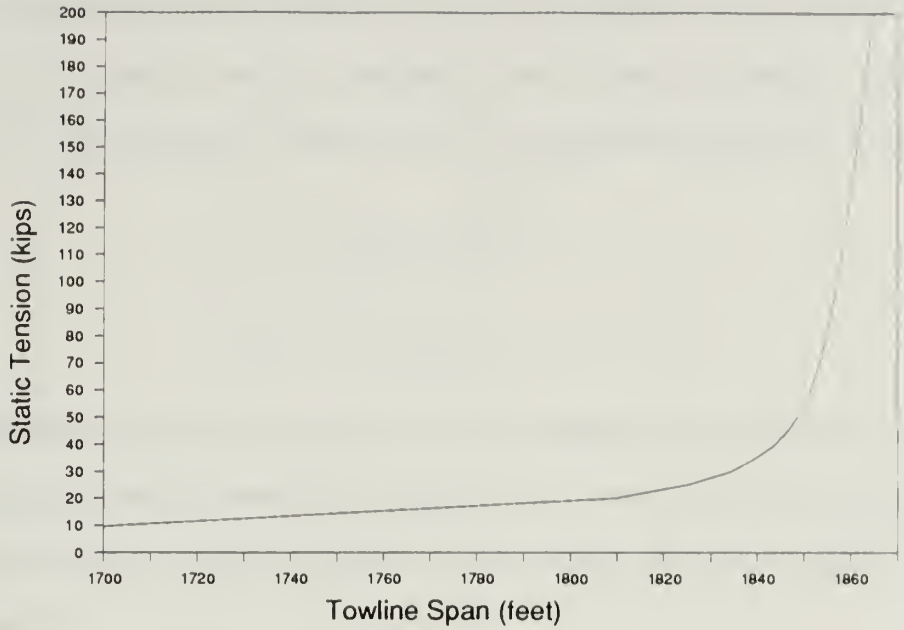

Figure 2.2. Towline Tension vs. Span

2.3.2 Polynomial Approximation

The fundamental parameters used to describe the towline are the elongation (ξ) and its time derivative ($\dot{\xi}$), where the elongation of the towline is defined as the tangential motions at both ends. Based on a series of time domain simulations of towline behavior using equations (2.1) and (2.2) (confirmed by experimental results [2]), the nonlinear tension can be accurately represented as a polynomial approximation of the form

$$\tilde{T}_{NL}(\xi,\dot{\xi}) = \sum_{m=0}^{3}\sum_{n=0}^{3} a_{mn}\xi^{m}(t)\dot{\xi}^{n}(t)$$

$$\text{with:} \quad \begin{cases} m+n \le 3 \\ (m,n) \ne (0,0) \end{cases}$$

The time simulations are performed for the desired towline geometries and static tensions, forcing the towline sinusoidally at various frequencies and amplitudes over the range

of frequencies and amplitudes of interest in real towing situations. From these N simulations, typically several hundred, N triplets of elongation, rate of change of elongation, and dynamic tension $(\xi_i, \dot{\xi}_i, \bar{T}_i)$ are generated. Next, the set of N overspecified equations is formed as

$$\sum_{m=0}^{3}\sum_{n=0}^{3} a_{mn}\xi^m\dot{\xi}^n = \bar{T}_i \tag{2.3}$$

$$\text{with: } \begin{cases} m+n \le 3 \\ (m,n) \ne (0,0) \end{cases} \quad i = 1, \ldots, N$$

and the coefficients a_{mn} are determined such that the weighted mean square error is minimized. Weighting is used to ensure the resulting towline model is most accurate for the larger tensions which are of greater concern for ship motions and towline safety. The weighting used is

$$\frac{1}{\omega_i(\Xi_i/D)}\frac{\bar{T}_i}{\bar{\bar{T}}}$$

where Ξ_i is the elongation amplitude, ω_i is the circular frequency of the excitation, and D is the cable diameter.

2.4 Towing System Model

A tug and tow at sea comprise a twelve-degree-of-freedom system, with each ship being driven in its own six degrees of freedom by both ocean waves and by the dynamic tension of the connecting towline. Figure 2.3 shows the geometry of such a towing system.

The statistics of the motions of a single ship in a seaway are well modeled by linear seakeeping theory in terms of the ship motion frequency responses to wave surface elevation, using a stochastic description of ocean waves, such as a Pierson-Moskowitz spectrum (see Figure 2.4), as input. Until recently, however, there has been little analysis of the more complex problem presented by one ship towing another [2].

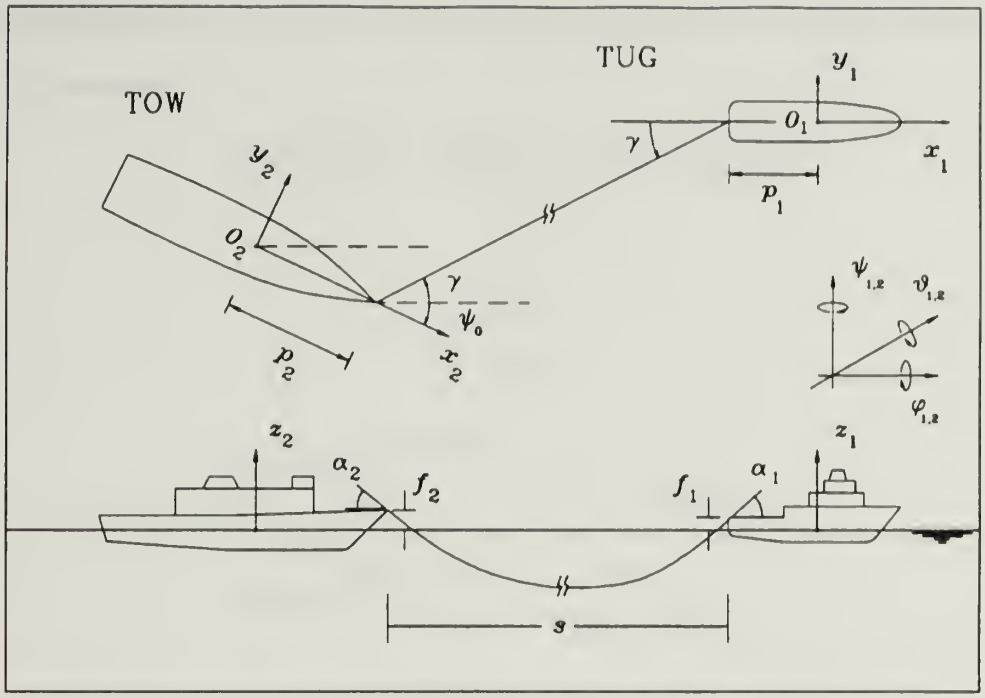

Figure 2.3. Towing Geometry

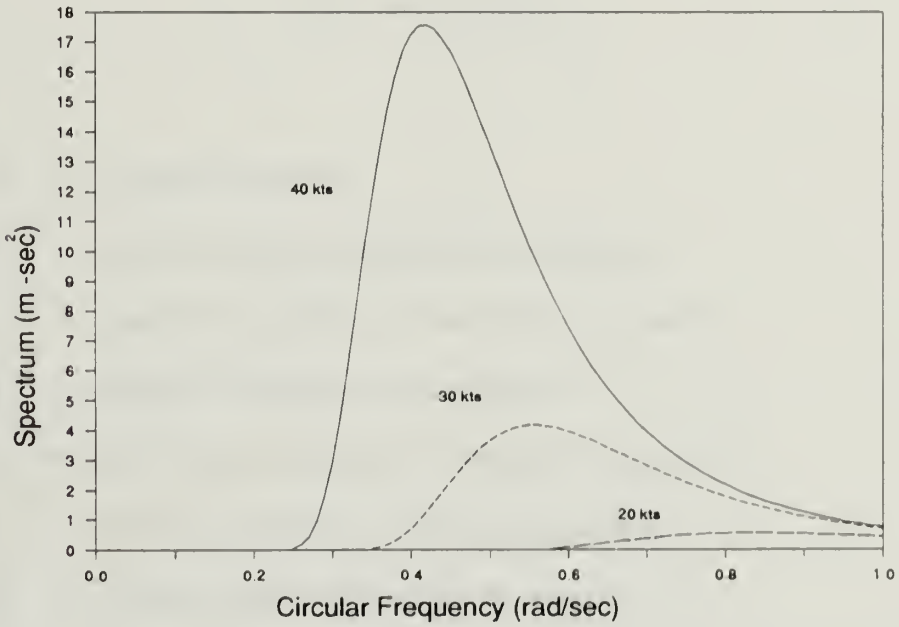

Figure 2.4. Pierson-Moskowitz Sea Spectrum

The equations of motion of the twelve-degree-of-freedom system can be written in the frequency domain as

$$[-\omega_e^2 \mathbf{M}_s(\omega_e) + i\omega_e \mathbf{B}_s(\omega_e) + \mathbf{C}_s]\mathbf{X}_s = \mathbf{F}_s(\omega_e) \tag{2.4}$$

where

$$\mathbf{X}_s = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{bmatrix}_{12x1}$$

$$\mathbf{M}_s(\omega_e) = \begin{bmatrix} \mathbf{M}_1 + \mathbf{A}_1(\omega_e) & 0 \\ 0 & \mathbf{M}_2 + \mathbf{A}_2(\omega_e) \end{bmatrix}_{12x12}$$

$$\mathbf{B}_s(\omega_e) = \begin{bmatrix} \mathbf{B}_1(\omega_e) - \mathbf{D}_{11} & -\mathbf{D}_{12} \\ -\mathbf{D}_{21} & \mathbf{B}_2(\omega_e) - \mathbf{D}_{22} \end{bmatrix}_{12x12}$$

$$\mathbf{C}_s = \begin{bmatrix} \mathbf{C}_1 - \mathbf{K}_{11} & -\mathbf{K}_{12} \\ -\mathbf{K}_{21} & \mathbf{C}_2 - \mathbf{K}_{22} \end{bmatrix}_{12x12}$$

$$\mathbf{F}_s(\omega_e) = \begin{bmatrix} \mathbf{F}_1(\omega_e) \\ \mathbf{F}_2(\omega_e)e^{i\theta} \end{bmatrix}_{12x1}$$

and where

ω_e	=	encounter frequency
$\mathbf{M}_1, \mathbf{M}_2$	=	generalized mass matrices for tug and tow
$\mathbf{A}_1, \mathbf{A}_2$	=	generalized added mass matrices for tug and tow
$\mathbf{B}_1, \mathbf{B}_2$	=	damping coefficients for tug and tow
$\mathbf{C}_1, \mathbf{C}_2$	=	hydrostatic restoring force matrices for tug and tow
$\mathbf{F}_1, \mathbf{F}_2$	=	wave exciting force vectors for tug and tow
$\mathbf{B}_1, \mathbf{B}_2$	=	motion response vectors for tug and tow

θ is the phase of the wave at the center of gravity of the tow with respect to the phase of the wave at the center of gravity of the tug.

The effects of the towline on the ship motions are found by using the theory of equivalent linearization [1], which assumes a linear relationship between the towline tension and its extension and time derivative of extension. Using this theory, the total towline tension, T_T, the sum of static and dynamic tensions, is given by

$$T_T = \overline{T} + k_{eq}\xi + b_{eq}\dot{\xi}$$

The linearized unsteady extension is

$$\xi = \mathbf{S}_1^T \cdot \mathbf{X}_1 + \mathbf{S}_2^T \cdot \mathbf{X}_2$$

where $\mathbf{S}_1$ and $\mathbf{S}_2$ are given in terms of the towing geometry (Figure 2.3) as

$$\mathbf{S}_1 = \begin{bmatrix} \cos\alpha_1 \cos\gamma \\ \cos\alpha_1 \sin\gamma \\ \sin\alpha_1 \\ -f_1 \cos\alpha_1 \sin\gamma \\ f_1 \cos\alpha_1 \cos\gamma + p_1 \sin\alpha_1 \\ -p_1 \cos\alpha_1 \sin\gamma \end{bmatrix}$$

$$\mathbf{S}_2 = \begin{bmatrix} -\cos\alpha_2 \cos(\gamma+\psi_0) \\ -\cos\alpha_2 \sin(\gamma+\psi_0) \\ \sin\alpha_2 \\ f_2 \cos\alpha_2 \sin(\gamma+\psi_0) \\ -f_2 \cos\alpha_2 \cos(\gamma+\psi_0) - p_2 \sin\alpha_2 \\ -p_2 \cos\alpha_2 \sin(\gamma+\psi_0) \end{bmatrix}$$

The unsteady forces exerted on the tug and the tow by the towline can be written, in the frequency domain, as

$$\mathbf{F}_{H_1} = \mathbf{K}_{11}\mathbf{X}_1 + \mathbf{K}_{12}\mathbf{X}_2 + \mathbf{D}_{11}\dot{\mathbf{X}}_1 + \mathbf{D}_{12}\dot{\mathbf{X}}_2$$

$$\mathbf{F}_{H_2} = \mathbf{K}_{12}\mathbf{X}_1 + \mathbf{K}_{22}\mathbf{X}_2 + \mathbf{D}_{21}\dot{\mathbf{X}}_1 + \mathbf{D}_{22}\dot{\mathbf{X}}_2$$

given in terms of the restoring force matrices **K** and the damping matrices **D**. Expressions for the **K**'s and **D**'s are given by Frimm [2].

Solving equation (2.4) gives the frequency domain ship motion vector in terms of the wave exciting force vector as

$$\mathbf{X}_s(\omega_e) = [-\omega_e^2\mathbf{M}_s(\omega_e) + i\omega_e\mathbf{B}_s(\omega_e) + \mathbf{C}_s]^{-1}\mathbf{F}_s(\omega_e)$$

where again both $\mathbf{X}_s$ and $\mathbf{F}_s$ are for the complete twelve-degree-of-freedom system.

The wave excitation forces can be written as

$$\mathbf{F}_s(\omega_e, \beta) = W(\omega_e, \beta)\mathbf{H}_{WF}(\omega_e, \beta)$$

where ω_e is the encounter frequency of the wave, β is the propagation angle of the wave with respect to the course of the tug (see Figure 2.5), $W(\omega_e, \beta)$ is the wave amplitude, and $\mathbf{H}_{WF}(\omega_e, \beta)$ is the transfer function from wave amplitude to tug and tow wave forces.

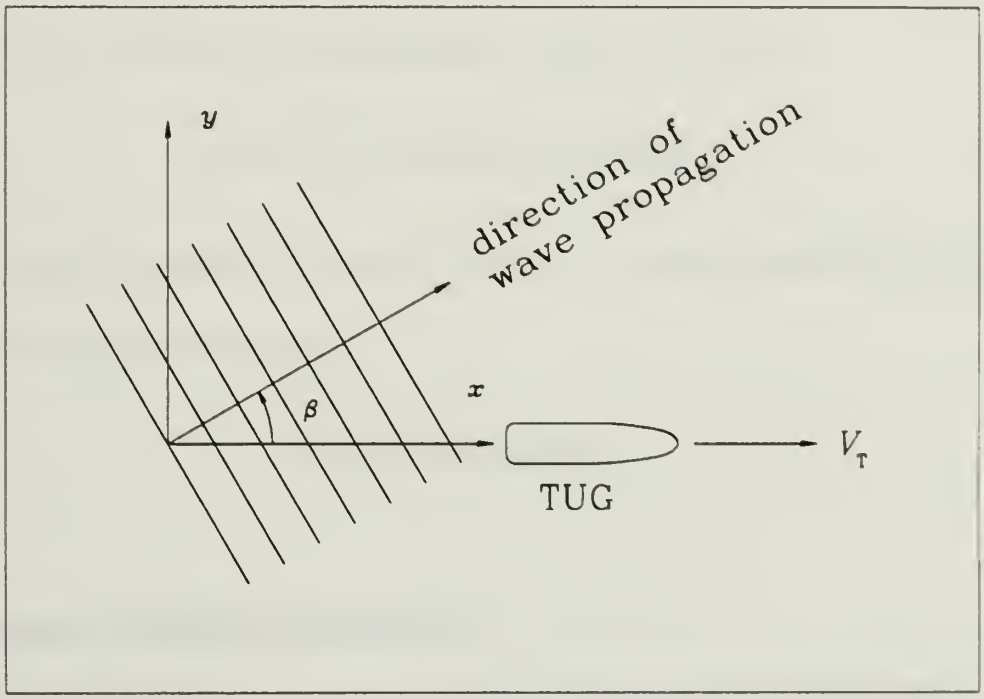

Figure 2.5. Wave Angle Coordinate System

The frequency domain towline extension depends on the wave amplitude and direction as

$$\Xi(\omega_e, \beta) = \mathbf{S}^T[-\omega_e^2\mathbf{M}_s(\omega_e) + i\omega_e\mathbf{B}_s(\omega_e) + \mathbf{C}_s]^{-1}\mathbf{H}_{WF}(\omega_e, \beta)W(\omega_e, \beta)$$

where $\mathbf{S}^T$ is the transpose of S, given by

$$\mathbf{S}^T = [\mathbf{S}_1^T \mathbf{S}_2^T]$$

The directional transfer function from wave amplitude to towline extension can thus be expressed as

$$H_{W\xi}(\omega_e, \beta) = \mathbf{S}^T[-\omega_e^2\mathbf{M}_s(\omega_e) + i\omega_e\mathbf{B}_s(\omega_e) + \mathbf{C}_s]^{-1}\mathbf{H}_{WF}(\omega_e, \beta)$$

and for any directional encounter frequency wave power density spectrum $S_{WW}(\omega_e, \beta)$, the linearized towline extension power density spectrum, $S_{\xi\xi}(\omega_e)$, is given by

$$S_{\xi\xi}(\omega_e) = \int_0^{2\pi} S_{WW}(\omega_e, \beta)\,|H_{W\xi}(\omega_e, \beta)|^2 d\beta \tag{2.5}$$

We will need the moments of $S_{\xi\xi}(\omega_e)$, $m_{n_\xi}, n = 0, 2, \ldots$, in the analysis of extreme tension statistics. These moments are given by

$$m_{n_\xi} = \int_0^{\infty} \omega_e^n S_{\xi\xi}(\omega_e) d\omega_e$$

2.5 Extreme Tension Statistics

Since the sea waves, which are a random process, are the driving excitations in the dynamic motions of the tug and tow, and thus in the magnitude of the dynamic towline tension, the towline tension will also be a random process. Thus, the best way to predict towline tension is to use a statistical approach.

The most useful statistic is the extreme tension that has a prescribed probability of not occurring during a given time. Such a statistic is called a "short term extremal statistic", to which we turn next.

2.5.1 Short Term Extremal Statistics

Suppose the dynamic tension $\tilde{T}(t)$ is linear, and thus a gaussian random process. We define ${}_{\alpha}T_{\Delta t}$ as the tension value that has a probability α of being exceeded in a period of Δt seconds. For example, ${}_{0.001}T_{86400}$ is the value of $\tilde{T}$ that has a probability of 0.001 of being exceeded in 86,400 seconds (one day).

The theory of short term extremal statistics of large extremes of gaussian random processes gives

$$ {}_{\alpha}T_{\Delta t} = \sqrt{2m_{0_T} \ln \left(\frac{t}{2\pi\alpha} \sqrt{m_{2_T}/m_{0_T}} \right)} $$

where m_{0_T} and m_{2_T} are the zeroth and second moments of $S_{TT}(\omega)$, the power density spectrum of the random process $\tilde{T}(t)$.

While this result is valid for finding the statistics of extremes for ship motions and the towline extension, it is not valid for the extreme tensions of wire rope towlines. This is because the highly nonlinear relationship between towline extension and tension (Figure 2.2) makes the dynamic tension, $\tilde{T}(t)$, a very non-gaussian random process.

The maxima of the total towline tension is formed by the maxima of the dynamic tension $\tilde{T}$ added to the static tension $\overline{T}$. Since $\overline{T}$ is constant, to evaluate tension extremes we need consider only $\tilde{T}$. A theory for extremes which allows inclusion of the nonlinear behavior of the towline [1] follows.

A cycle of $\tilde{T}$ is defined as the behavior of $\tilde{T}$ from a time that it is zero and increasing (a positive zero upcrossing) until the time of the next positive zero upcrossing. The maximum value, or peak, reached by $\tilde{T}$ during one cycle is a random variable, Y. The probability distribution function for Y, $F_Y(Y_0)$, is the probability that Y is less than Y_0.

For a large number of cycles, n, and a value Y_0 much larger than the average peak, the largest peak during this period is designated Y_n. The probability that Y_n is less than Y_0, $F_{Y_n}(Y_0)$, is approximated as

$$F_{Y_n}(Y_0) = [F_Y(Y_0)]^n$$

This approximation assumes that because Y_0 is large, occurrences of it being exceeded are well separated in time so they can be considered statistically independent.

The value of Y_n that has a probability α of being exceeded is ${}_\alpha Y_n$, and so

$$[F_Y({}_\alpha Y_n)]^n = 1 - \alpha \quad \text{or} \quad F_Y({}_\alpha Y_n) = (1-\alpha)^{1/n}$$

If the function $F_Y(Y_0)$ is known, then ${}_\alpha Y_n$ can be found as

$${}_\alpha Y_n = F_Y^{-1}[\,(1-\alpha)^{1/n}\,]$$

Let ${}_\alpha T_n$ be the random value of the largest tension peak occurring in a time interval Δt whose probability of being exceeded is α. The average number of cycles per second of T is called $N(0)$. Thus, ${}_\alpha T_n$ is related to ${}_\alpha Y_n$ by $n = N(0)\Delta t$ and so

$${}_\alpha T_{\Delta t} = F_Y^{-1}\,[\,(1-\alpha)^{1/N(0)\Delta t}\,] \tag{2.6}$$

${}_\alpha Y_n$ and ${}_\alpha T_n$ are short term extremal statistics of the tension peaks. To find them we must determine $F_Y(Y_0)$, the probability distribution function (pdf) for the value of an arbitrary tension peak.

The frequency of upcrossings by $\bar{T}$ of any dynamic tension level Y_0 is called $N(Y_0)$, which is given by

$$N(Y_0) = \int_0^\infty \dot{T} f(Y_0, \dot{T}) d\dot{T} \tag{2.7}$$

where $f(\bar{T}, \dot{T})$ is the joint pdf for the tension and its time-derivative. The frequency of zero upcrossings is

$$N(0) = \int_0^\infty \dot{T} f(0, \dot{T}) d\dot{T}$$

These frequencies are related to the pdf of an arbitrary tension peak through

$$F_Y(Y_0) = 1 - \frac{N(Y_0)}{N(0)} \tag{2.8}$$

The approach is to find the level crossing frequencies, $N(Y_0)$, then use equation (2.8) to find the pdf for the peak amplitudes, $F_Y(Y_0)$, and then use equation (2.6) to find the desired extremal statistics.

The details of this analysis are given by Frimm [1], with the result that equation (2.7) is integrated directly in the $(\xi, \dot{\xi})$ plane, after a suitable transformation of variables to

$$N(Y_0) = \int_{-\infty}^{\infty} \left[\int_{s_0}^{s_f} \dot{T}(Y_0, s, \ddot{\xi}) u(\dot{T}(Y_0, s, \ddot{\xi})) p_\xi(Y_0, s, \ddot{\xi}) \frac{1}{|\bar{\nabla}\bar{T}(s)|} ds \right] d\ddot{\xi}$$

where $s \equiv (\xi_0, \dot{\xi}_0)$ are the points along the constant tension curve $\bar{T} = Y_0$ in the $(\xi, \dot{\xi})$ plane, s, and s_f are the initial and final values of s, $u(...)$ is the unit step function, and $p_\xi(...)$ is the joint pdf of the cable extension. This last is given by

$$p_\xi(\xi, \dot{\xi}, \ddot{\xi}) = \frac{1}{(2\pi)^{3/2}\sqrt{m_{2_\xi}\Delta}} \exp\left[-\frac{1}{2\Delta}\left(m_{4_\xi}\xi^2 + \frac{\Delta}{m_{2_\xi}}\dot{\xi}^2 + 2m_{2_\xi}\xi\ddot{\xi} \right) \right]$$

where $\Delta \equiv m_{0_\xi} m_{4_\xi} - m_{2_\xi}^2$, and $m_{0_\xi}, m_{2_\xi}, m_{4_\xi}$ are the extension spectral moments determined from equation (2.5).

2.5.2 Computational Method

Figure 2.6 shows the overall scheme for computing the extreme tension statistics. First, the towline governing equations are solved to produce a set of time simulations of cable behavior. From these, equation (2.3) is solved for the a_{mn}'s for each towline condition. With these, the twelve-degree-of-freedom motion equations are solved iteratively, using the equivalent linear towline model to predict ship motions. Once convergence is obtained, the resulting towline extension spectrum is used to calculate the "extreme tension", T_{ext}, which is defined as

$$T_{ext} = {}_{0.001}\tilde{T}_{8640.0} + \overline{T}$$

and $\overline{T}$ is the mean static tension.

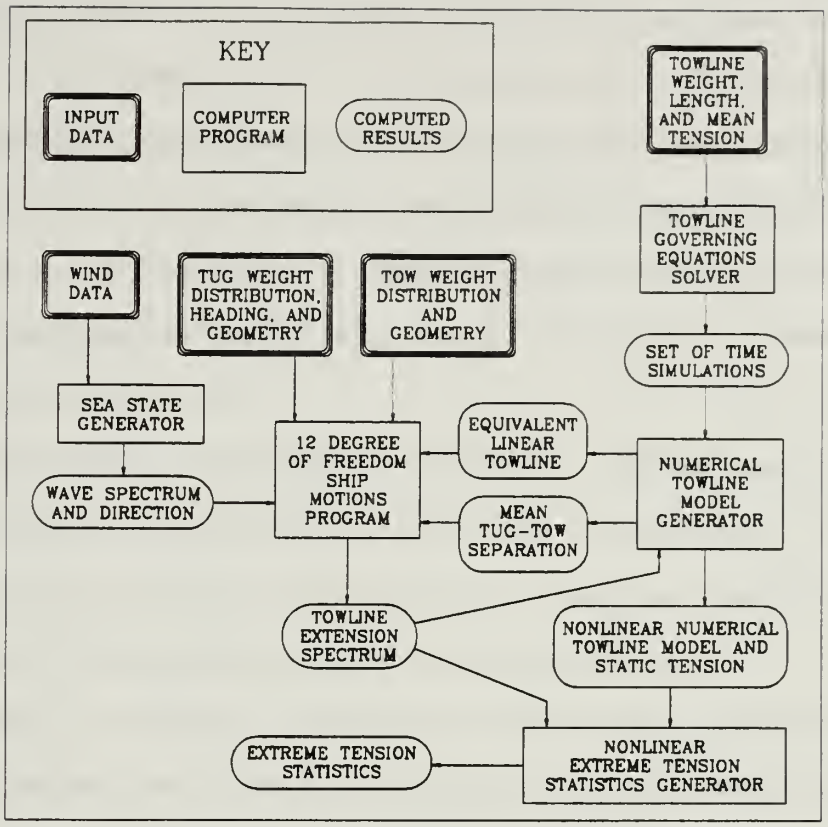

Figure 2.6. Data Flow for Computational Scheme

2.5.3 Data Base Development

Ideally, this method for predicting the nonlinear extreme tensions would be used directly by tow planners and tug operators. However, the computational requirements are currently beyond the resources available to these potential users. Consequently, a more practical alternative was developed to bring the results of this analysis to the fleet.

Since the extreme tension depends on tug and tow geometries and weight distributions, towline characteristics and length, mean tension, towing speed, sea state, and towing direction with respect to the waves, a data base of extreme tensions was developed for various combinations of these parameters. Table 2.1 lists the particular values used in generating the data base.

The resulting extreme tensions were plotted against the mean tension and compared against a set of 100 "standard curves" [2]. Based on that comparison, a table was developed showing the appropriate standard curve to use for the given combination of towing parameters. The curve selected is the one which minimizes the error between it and the tensions computed from the non-linear theory. A weighted least squares method was used such that the curve selected minimized the error near extremes corresponding to 70 percent of the breaking strength of the tow wire.

Although the range of towed vessels in the data base of extreme tensions covers most of the vessels in the Navy, the very largest, aircraft carriers, were omitted. Towing an aircraft carrier is a rare event, but should the need arise, it is important that Commanding Officers and Masters be able to predict their ability to safely tow the vessel.

Consequently, a representative aircraft carrier, CVN 65, with displacement of 85,000 long tons, was chosen to be included in the data base. The tables of standard curve choices that resulted are given in Appendix A. To facilitate use by Fleet personnel, the wave angle β is replaced by a heading angle, where

$$\text{heading angle} = \text{wave angle} + 180^{\circ}.$$

Tugs:	ARS 38	ARS 50/ATS 1	T-ATF 166
Towline dia:	2.00 in	2.25 in	2.25 in
Displacement:	1900 LT	3200 LT	2260 LT

Tows	Displacement
YRBM Berthing barge	650 LT
FFG 1 Frigate	3200 LT
DD 963 Destroyer	6700 LT
AE 26 Ammunition Ship	20,000 LT
LHA 1 Assault Carrier	40,000 LT

Wind Speed (kts)	15	20	25	30	
Tow Speed (kts)	3.0	6.0	9.0		
Towline Length (ft)	1000	1200	1500	1800	2100
Mean Tension (lbs)	10,000	20,000	40,000	80,000	120,000
Wave Angle (deg)	0	60	120	180	

Table 2.1. Data Base Towing Parameters.

Chapter 3

Resistance Prediction

In order to use the extreme tension results of the previous chapter, the mean towline tension must be measured or estimated. In this chapter, I discuss methods for predicting both the resistance of the towed vessel and the resistance of the towing hawser.

3.1 Tow Resistance

The resistance of a vessel in a seaway at low speeds is not well understood, particularly because the added resistance due to waves at low speeds is a poorly understood phenomenon. Nevertheless, the U.S. Navy Towing Manual [3] gives largely empirical procedures for estimating the resistance of vessels at towing speeds. These have produced reasonably useful results over the years, and they are adopted herein for lack of any better, practically useful methods.

The total tow resistance, R_T, is assumed to be made up of four components, and can be written as

$$R_T = R_H + R_S + R_W + R_P$$

where

R_H = calm water hull resistance

R_W = wind resistance

R_S = added resistance due to waves

R_P = resistance due to the tow's propellers

The Towing Manual distinguishes between three different types of towed vessels: self-propelled ships, floating drydocks, and barges. Somewhat different methods are used to compute the separate components of resistance for each type of vessel, and we will look at each in turn.

3.1.1 Hull resistance

3.1.1.1 Self-propelled ships

The calm water resistance for self-propelled ships is given (in pounds) as

$$R_H = 1.25 \times \left(\frac{R_H}{\Delta} \right) \times \Delta$$

where R_H/Δ is given as a function of towing speed in Figure 3.1 (reproduced from [3]), and Δ is the displacement in long tons. The appropriate curve to be used in Figure 3.1 is given in Table G-2 of the Towing Manual, which lists data for 141 types of Naval vessels. A twenty-five percent margin is included to account for hull fouling and other uncertainties.

3.1.1.2 Drydocks

For floating drydocks, the calm water resistance in pounds is given by

$$R_H = f_1 \times S \times (V_T/6)^2$$

where

f_1 = a coefficient depending on the degree of fouling of the hull

S = wetted surface area of the drydock

V_T = tow speed in knots

The values of S and f_1 are obtained from Table 3.1 (reproduced from [3]).

3.1.1.3 Barges

The method for barges is precisely the same as for drydocks, except that the wetted surface area S is given by

$$S = (\text{length}) \times (\text{width}) + (\text{perimeter}) \times (\text{draft})$$

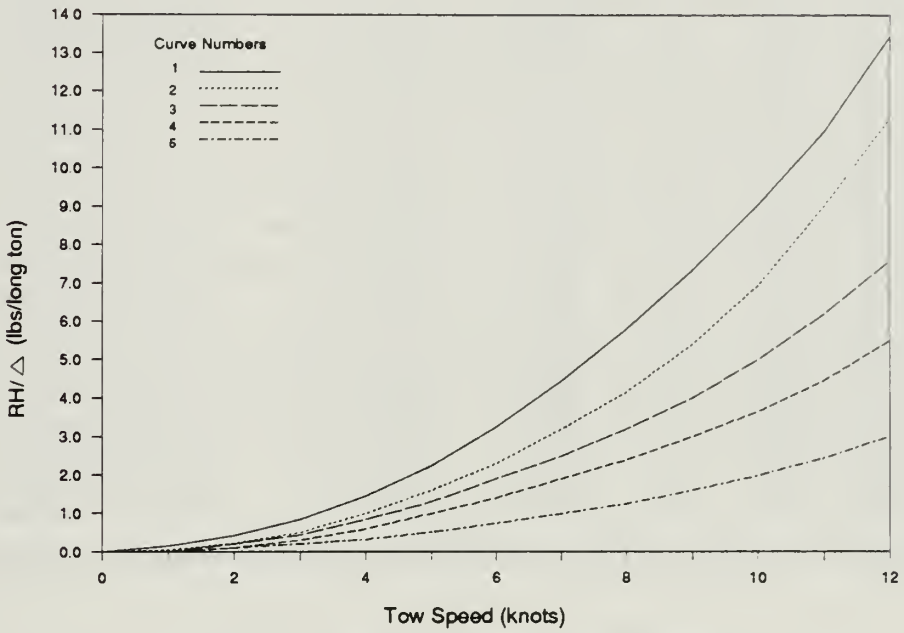

Figure 3.1. Hull Resistance Curves for Self-Propelled Ships

Drydock Class		f1	S	f2	B	f3	C
AFDB-1	Section 256' x 80'	0.45-0.8	23,000	0.3	720	0.70	3,800
AFDB-4	Section 240' x 101'	0.45-0.8	26,000	0.3	900	0.70	4,500
AFDM-1	Section 496' x 116'	0.45-0.8	50,000	0.6	750	0.67	7,000
AFDM-3	Section 488' x 124'	0.45-0.8	52,000	0.6	800	0.67	7,800
ARD-1	Section 390' x 60'	0.45-0.8	20,000	0.2	250	0.61	2,000
ARD-2	Section 486' x 71'	0.45-0.8	34,000	0.2	370	0.61	3,700
ARD-12	Section 492' x 81'	0.45-0.8	40,000	0.2	480	0.61	4,400
AFDL-1	Section 200' x 64'	0.45-0.8	13,000	0.4	220	0.70	1,400
AFDL-7	Section 288' x 64'	0.45-0.8	19,000	0.4	210	0.70	1,500
AFDL-35	Section 389' x 84'	0.45-0.8	38,000	0.3	780	0.67	1,900
AFDL-47	Section 448' x 97'	0.45-0.8	46,000	0.5	420	0.70	2,500
AFDL-48	Section 400' x 96'	0.45-0.8	48,000	0.4	1,350	0.70	2,540
YFD-7	Section 488' x 124'	0.45-0.8	52,000	0.6	800	0.67	7,800
YFD-68 to 71	Section 474' x 118'	0.45-0.8	48,000	0.6	750	0.67	7,300

Table 3.1. Drydock Towing Coefficients

3.1.2 Wave Resistance

3.1.2.1 Self-propelled ships

For self-propelled ships, the added resistance due to waves is estimated using the curves in Figure 3.2, reproduced from the Towing Manual, based on the hull form. The entering argument in these curves is the wave height in feet; the user is expected to use observed wave height, or to infer wave height based on wind force (Beaufort number) from data such as that in Table 3.2. The latter approach does not lend itself well to a digital application, since a Beaufort number does not correspond to a unique wind speed or a unique wave height.

A more attractive alternative for computer implementation is to associate a unique wave height with a given wind speed. This can be readily accomplished using a spectral description of wave energy, such as the Pierson-Moskowitz spectrum (Figure 2.4).

The Pierson-Moskowitz spectrum is an example of a general two-parameter sea spectrum, which is described by

$$S(\omega) = (\alpha g^2/\omega^5) \exp(-\beta(g/V_w\omega)^4)$$

where

$S(\omega)$ = frequency spectrum, $m^2 \cdot sec$

ω = wave frequency, radians/sec

α , β = nondimensional parameters

g = gravitaitonal acceleration constant, m/sec^2

V_w = wind velocity at 19.5 m above free surface, m/sec

For the Pierson-Moskowitz spectrum, the nondimensional parameters take the values

$$\alpha = 8.1 \times 10^{-3} \qquad \beta = 0.74$$

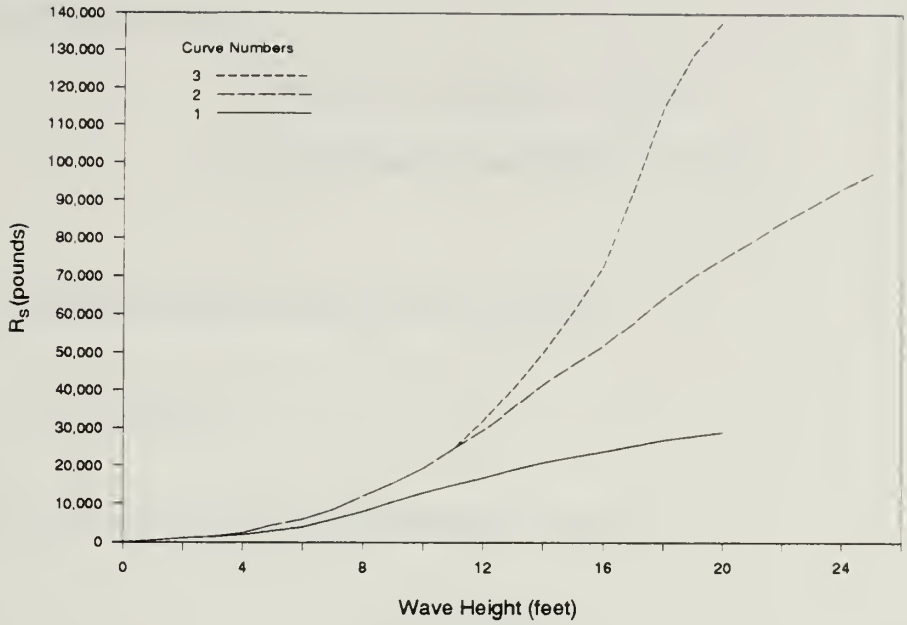

Figure 3.2. Added Resistance Curves for Self-Propelled Ships

Beaufort No.	Wind Speed (kts)	Description	$H_{1/3}$ (feet)	Ave. Wave Length (feet)	Minimum Duration (Hours)
0	< 1	Calm			
1	1-3	Light air	< 1	10 in	18 min
2	4-6	Light breeze	< 1	6.7 ft	39 min
3	7-10	Gentle breeze	< 1	20-27	1.7-2.4 (hours)
4	11-16	Moderate breeze	2.9-4.6	52-71	4.8-6.6
5	17-21	Fresh breeze	6.1-8.0	90-111	8.3-10.0
6	22-27	Strong breeze	10-15	134-188	12-17
7	28-33	Moderate gale	18-26	212-285	20-27
8	34-40	Fresh gale	30-45	322-444	30-42
9	41-47	Strong gale	50-64	492-590	47-57
10	48-55	Storm	71-95	650-810	63-81

Table 3.2. Beaufort Scale (From U.S. Navy Towing Manual)

Wave heights can be determined from this spectrum in terms of spectral moments, with the result that (Newman [5])

$$\overline{H} \equiv \text{average wave height} = \sqrt{2\,\pi\, m_0}$$

$$\overline{H}_{1/3} \equiv \text{significant } 1/3 \text{ waveheight} = 4\,\sqrt{m_0}$$

where

$m_0 \equiv$ zeroth moment $\equiv$ total energy in spectrum

$$= \int_0^\infty S(\omega)\, d\omega$$

Computing m_0 for the general two-parameter spectrum gives

$$m_0 = \frac{1}{4}\left(\frac{\alpha}{\beta}\right)\left(\frac{V_W^4}{g^2}\right)$$

Selecting $\overline{H}_{1/3}$ as a more relevant measure of wave height than the average wave height when considering towing dynamics, we have

$$\overline{H}_{1/3} = 2\,(V_W^2/g)\,\sqrt{\alpha/\beta}$$

$$= k\, V_W^2$$

giving wave height as a function of wind speed alone. Using English units gives

$$\overline{H}_{1/3} = 0.0186 \times V_W^2$$

with V_W in knots and $\overline{H}_{1/3}$ in feet.

Given the wave height, the added resistance is found directly by entering Figure 3.2 using the appropriate curve indicated by Table G-2 [3].

3.1.2.2 Drydocks and Barges

For drydocks and barges, an empirical rule is used to compute the added resistance due to waves:

$$R_{wave} = 2.85 \times B \times f_2 \times V^2 \times K$$

where

R_{wave} = resistance, lbs

B = cross-sectional area below the waterline, ft^2

f_2 = coefficient depending upon bow and stern configuration

V = speed of tow, kts

K = 1.2, a 20 percent allowance for rough water and eddies

Table 3.2 gives the values of B and f_2 for the Navy's floating drydocks. For barges, B = (beam) × (draft), $f_2 = 0.2$ for typical rake-ended barges or blunt ship-ended barges, and $f_2 = 0.5$ for square-ended barges.

3.1.3 Wind Resistance

3.1.3.1 Self-propelled ships

The wind resistance for self-propelled ships is given (in pounds) as

$$R_W = 0.00506 \times C_w \times A_T \times K \times V_R^2$$

where C_w is a wind coefficient, found in Table G-2 of the Towing Manual, A_T is the frontal projected area of the ship, also found in Table G-2, K is a heading coefficient, based on the relative wind direction, and V_R is the relative wind speed in knots.

The heading coefficient K is given by

$$K = \begin{cases} 1.0 & 0 \le \gamma < 20 \\ 1.2 & 20 \le \gamma < 40 \\ 0.4 & 40 \le \gamma \le 90 \end{cases}$$

where γ is the relative wind direction in degrees, with zero dead ahead.

The relative wind speed is found as

$$V_R^2 = V_T^2 + V_W^2 + 2V_T V_W \cos(180 - \gamma)$$

where

V_T = tow speed, knots

V_W = wind speed, knots

3.1.3.2 Drydocks and Barges

The wind resistance for floating drydocks is given (in pounds) as

$$R_W = 0.004 \times C \times f_3 \times (V_W + V_T)^2$$

where

C = cross-sectional area above waterline, sq ft

f_3 = coefficient depending on vessel shape above waterline

and V_T, V_W are as before. C and f_3 are given by Table 3.1.

For barges, the same expression is used as for drydocks, with

$$C = (\text{freeboard}) \times (\text{beam}) + (\text{height})_{\text{DECKHOUSE}} \times (\text{width})_{\text{DECKHOUSE}}$$

and $f_3 = 0.60$.

3.1.4 Propeller Resistance

The resistance of a ship's propeller that is locked in place is given by

$$R_P = 3.737 \times A_P \times V_T^2$$

where A_P is the projected area of the propeller, found in Table G-2 of the Towing Manual, and V_T is the tow speed in knots, as before. If instead the propeller is allowed to trail, or freewheel, this resistance is reduced by half; if the propeller is removed, the propeller resistance is zero.

Floating drydocks and barges of the types considered here do not have propellers, so for them the propeller resistance is zero.

3.2 Towline Resistance

The Towing Manual gives two methods for estimating the resistance of a towing hawser. The first is to take ten percent of the total resistance of the tow; the second involves interpolating a table of data given in the Towing Manual. Neither method is particularly useful: the first overstates the resistance in most cases, while the second is difficult to apply.

An improved estimate can be made by analyzing the hydrodynamics of the hawser. Much of the analysis here follows Berteaux [6]. It is assumed throughout that the hawser is perfectly flexible, that it does not stretch, and that it is symmetrical about its midpoint. The geometry of a towing hawser is shown in Figure 3.3.

Although the towline is moving through the water, we consider the equivalent problem of a static cable in a constant horizontal current.

The resistance of the hawser has two components, one due to normal pressure drag, and the second due to tangential frictional resistance as shown in Figure 3.4. The normal contribution is given by

$$D = \frac{1}{2}\rho\, C_{DN}\, d\, (V_N)^2$$

per unit length of cable, and where

D = normal drag component

ρ = density of water (slugs/ft^3)

C_{DN} = normal drag coefficient

d = hawser diameter (ft)

V_N = normal velocity component

Since $V_N = V \sin\phi$, the normal drag component becomes

$$D = \frac{1}{2}\rho\, C_{DN}\, d\, V^2 \sin^2\phi$$

Similarly, the tangential contribution per unit length is given by

$$F = \frac{1}{2}\rho\, C_{DT}\, \pi\, d\, (V_T)^2$$

or

$$F = \frac{1}{2}\rho\, C_{DT}\, \pi\, d\, V^2 \cos^2\phi$$

where

F = tangential drag component

C_{DT} = tangential drag coefficient

and where $V_T = V \cos\phi$ is the tangential velocity component.

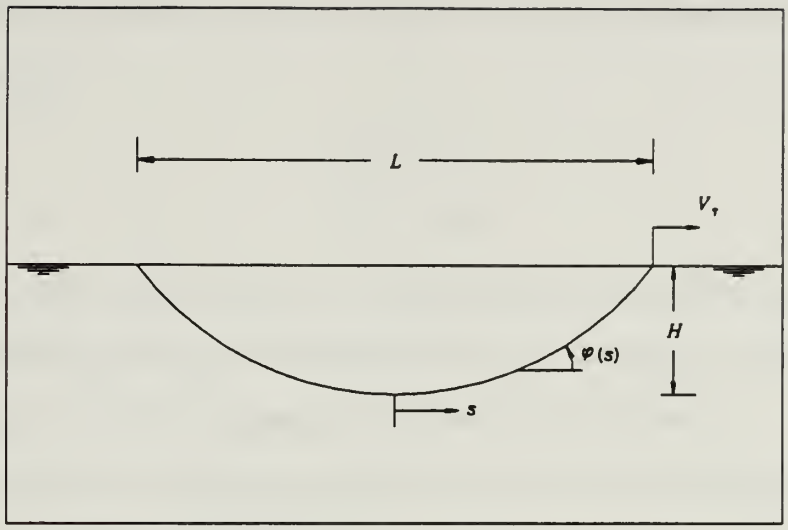

Figure 3.3. Towing Hawser Geometry

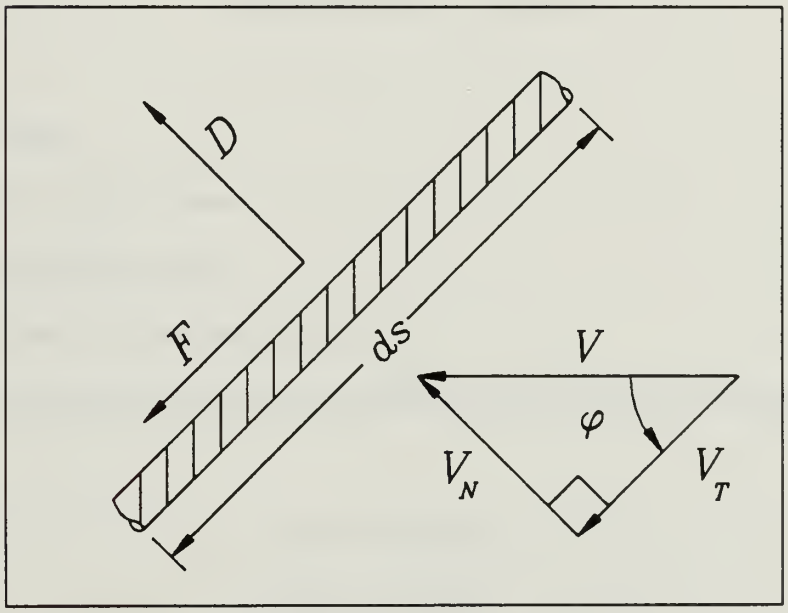

Figure 3.4. Tangential and Normal Forces on a Cable

The drag coefficients C_{DT} and C_{DN} both vary with Reynold's number, R_N. For the range of Reynold's numbers of interest in towing, these coefficients have the approximate values (Berteaux [6])

$$C_{DN} \approx 1.4$$

$$C_{DT} \approx 0.015$$

To find the total hawser resistance, we must integrate the horizontal components of the normal and tangential drag along the length of the hawser. Since both D and F are functions of the angle ϕ, we must first determine the geometry of the hawser.

Considering the static equilibrium of the hawser in the normal and tangential directions yields the following differential equations

$$dT = (P \sin\phi - F)\, ds$$

$$T d\phi = (D \sin^2\phi + P \cos\phi)\, ds$$

where

T = cable tension

P = weight of cable in water

ϕ = cable inclination angle

s = curvilinear coordinate along the cable

If we neglect the resistance of the hawser in determining its geometry, the equilibrium equations become

$$dT = P \sin\phi\, ds$$

$$T d\phi = P \cos\phi\, ds \tag{3.1}$$

Eliminating P and integrating over ϕ and T leads to the result

$$\phi = \tan^{-1}\left(\frac{P s}{T_0}\right)$$

where T_0 is the horizontal component of the static tension. This gives the inclination of the hawser as a function of its curvilinear distance from the midpoint.

Resolving D and F (Figure 3.4) into horizontal and vertical components gives

$$D_H = D \sin\phi$$

$$D_V = D \cos\phi$$

$$F_H = F \cos\phi$$

$$F_V = F \sin\phi$$

so the total horizontal resistance of the hawser per differential element is

$$dR = D_H + F_H = D \sin\phi + F \cos\phi$$

or,

$$dR = \left[\frac{1}{2}\rho\, C_{DN}\, d\, V^2 \sin^2\phi\right]\sin\phi$$

$$+\left[\frac{1}{2}\rho\, C_{DT}\, \pi\, d\, V^2 \cos^2\phi\right]\cos\phi$$

$$= \frac{1}{2}\rho\, d\, V^2 [C_{DN} \sin^3\phi + \pi\, C_{DT} \cos^3\phi]$$

Integrating this along the half-length of the hawser gives

$$R = \int_0^s \frac{1}{2}\rho\, d\, V^2 [C_{DN} \sin^3\phi + \pi\, C_{DT} \cos^3\phi]\, ds$$

which can be transformed into

$$R = \frac{1}{2}\rho\, d\, V^2 \left(\frac{T_0}{P}\right) \int_0^{\phi} \left\{ C_{DN} \frac{\sin^3\phi}{\cos^2\phi} + \pi\, C_{DT} \cos\phi \right\} d\phi \qquad (3.1)$$

where again $\phi = \tan^{-1}\left(\frac{Ps}{T_0}\right)$ and s is one half the total length of the hawser.

The total hawser resistance R_W is twice the value given by equation (3.1), and thus

$$R_W = \rho\, d\, V^2 \left(\frac{T_0}{P}\right)\left\{C_{DN}\left[\frac{\sin^4\phi}{\cos\phi} + \cos\phi(\sin^2\phi + 2) - 2\right] + \pi\, C_{DT} \sin\phi\right\} \qquad (3.2)$$

Neglecting the effects of the hawser drag on its geometry is not strictly valid, but when the depth of the hawser catenary, H, is small compared to the total span from end to end, L, as is usually the case in ocean towing (see Figure 3.4), the difference between the actual geometry and that of the assumed static catenary is small.

For a hawser moving through the water at a constant speed, as we have assumed throughout, equilibrium requires that the pull of the tug balance the resistance of the tow and the drag of the hawser. We have already estimated the resistance of the tow, and equation (3.2) gives an estimate for the hawser resistance. However, equation (3.2) requires the mean tension, T_0, as an argument, which is itself a function of the hawser resistance. This suggests an iterative scheme for finding the equilibrium configuration. For a given length of hawser, equation (3.2) is used to compute the hawser resistance at the given speed, using the tow resistance alone as a first estimate of T_0. The resulting hawser drag is added to the tow resistance to give a new estimate of the mean tension, which is then used to recompute the hawser resistance. This process is repeated until convergence is reached; that is, when successive estimates of hawser resistance differ by less than a predetermined amount.

Tables 3.3, 3.4, and 3.5 give representative samples of hawser resistance found by this method for several combinations of tow resistance and hawser scope at tow speeds of three, six, and nine knots.

We can see from these results that at higher values of ship resistance, the hawser drag is constant. This is because the hawser catenary has reached the limit of its ability to adjust its geometry, and thus the drag is due primarily to the tangential drag component.

V = 3 kts	Hawser Scope (ft)						
Ship Drag	1000	1200	1400	1600	1800	2000	2200
10,000	299	415	563	744	957	1199	1468
20,000	234	290	352	423	505	599	707
50,000	226	271	317	364	411	460	509
100,000	226	271	316	361	406	452	497

Table 3.3. Hawser Resistance (lbs) at 3 knots

V = 6 kts	Hawser Scope (ft)						
Ship Drag	1000	1200	1400	1600	1800	2000	2200
10,000	1137	1518	1967	2477	3039	3642	4279
20,000	934	1150	1388	1652	1946	2274	2637
50,000	904	1086	1269	1455	1643	1835	2031
100,000	903	1083	1264	1445	1626	1807	1989

Table 3.4. Hawser Resistance (lbs) at 6 knots

V = 9 kts	Hawser Scope (ft)						
Ship Drag	1000	1200	1400	1600	1800	2000	2200
10,000	2411	3108	3882	4720	5609	6538	7497
20,000	2090	2559	3062	3605	4192	4823	5498
50,000	2033	2442	2855	3271	3692	4119	4554
100,000	2031	2438	2844	3251	3658	4066	4475

Table 3.5. Hawser Resistance (lbs) at 9 knots

Chapter 4

Program Description

In this chapter I describe an interactive computer program, **TOWCALC**, which implements the theory and methods presented in Chapters 2 and 3. We begin with a brief overview, then proceed to a more detailed examination. Procedures for installing **TOWCALC** are given in Appendix C; Appendix D gives a complete program listing.

4.1 Overview

4.1.1 Purpose

The goal of this program is to provide tow planners and tug operators with a computational tool to aid in their decision making with regard to ocean towing. Since many of the intended users are only casual computer users, it is important the program be as transparent as possible. By this I mean the user should not be required to be especially computer literate, beyond a basic understanding of how to use an applications program. This requires the interface between the program and the user be as "friendly" as possible, which translates to a liberal use of menus and prompts, and extensive error checking of input.

4.1.2 Intended Users

The intended users of this program are primarily U.S. Navy personnel involved in the planning and execution of open ocean tows. This includes both staff personnel and shipboard operators. The most likely staffs to use the program would be Combat Support Squadron staffs, Military Sealift Command (MSC) personnel, and the office of the U.S. Navy

Supervisor of Salvage (SUPSALV). Afloat personnel would include the Commanding Officer and wardroom (other officers) of U.S. Navy towing salvage vessels, and the Masters and mates of MSC ocean going tugs.

4.1.3 Hardware Requirements

TOWCALC was developed for use on IBM and compatible micro-computers running under IBM PC-DOS™ or Microsoft MS-DOS™ versions 3.2 and higher, equipped with a hard disk drive, a graphics card, and a graphics monitor. Although it will work with both monochrome and color monitors, color is not used. A printer is also required.

Supported graphics devices include the IBM Color Graphics Array (CGA), Enhanced Graphics Array (EGA), and Hercules monochrome graphics. Better results are obtained on the higher resolution EGA and Hercules devices.

4.1.4 Programming Environment

TOWCALC is written in the C programming language, using the Microsoft Optimizing C Compiler™ Version 5.0; graphics routines use the Halo '88™ graphics kernel. C was chosen because it lends itself to structured programming, and because it provides complete control over all aspects of the computer. This is particularly important in developing a "friendly" user interface.

Several of the C functions used in **TOWCALC** are from a library written by Doerry [7]; the plotting functions are modified versions of those in a library by Milgram [8].

4.1.5 User Interface

The user interface is designed to make this program as simple to use as possible. Menus and prompts are used extensively, eliminating the need for the user to learn a command set. Some of the important elements of this interface are described below.

The basic display is a reverse video window in the center of the screen. Program options are selected from popup menus; cursor keys are used to move a highlighted bar from choice to choice, or the number of the selection may be pressed. Pressing the escape key generally moves up one level in the overall menu structure.

The user enters data by typing information on the keyboard, which is written to the screen in the appropriate location. Editing of data is accomplished by pressing the appropriate function key corresponding to the item, listed in a menu usually shown at the bottom of the display.

Error and warning messages are displayed in a popup window which appears in the center of the screen; the user presses any key and the message disappears, restoring the display beneath it. The action taken by the program in response to an error or a warning depends on the context of the situation.

When prompted to respond "yes" or "no", pressing the return or enter key (↵) generates a default response, usually "yes".

The interface functions (windows, popup menus, screen display) achieve a nearly instantaneous response by writing directly to video RAM (random access memory). They are based on a set of functions by Schildt [9].

4.1.6 Options

After an initial title screen (Figure 4.1) a popup menu appears (Figure 4.2) giving the five main options in this program:

1) Select Tug

2) Select Tow

3) Estimate Dynamic Tension

4) Print Report

5) QUIT

Figure 4.3 shows this menu structure schematically. A brief description of each option follows; a more detailed discussion will be given in later sections.

Select Tug. The user selects one of four U.S. Navy tugs, sets the towline, or hawser, length, and sets the size and length of the chain pendant which connects the hawser to the tow.

Select Tow. Here the user enters the characteristics of the towed vessel, the desired towing speed, and environmental parameters. The resistance of the tow and of the towline are estimated, using the methods of Chapter 3, and the ability of the chosen tug class to tow the vessel at the desired speed is evaluated. Three different types of vessels are considered: self-propelled ships, floating drydocks, and barges. Submarines are not included.

Estimate Dynamic Tension. In this option, the results of Chapter 2 are implemented to predict the extreme towline tension. The user may elect to use the estimated mean tension from the previous option, or to enter the actual mean tension as measured at sea. The user may choose to plot (on screen) the standard tension curve, to compute and plot the effects of changing tow speed, or to compute and plot the effects of changing towline length. These last two options give the user an immediate look at how tow speed and towline length affect extreme tension.

Print Report. This option produces a file containing the most recent data and results. If the user so chooses, a printout is made of this file.

QUIT. This option terminates program execution and returns control to the operating system.

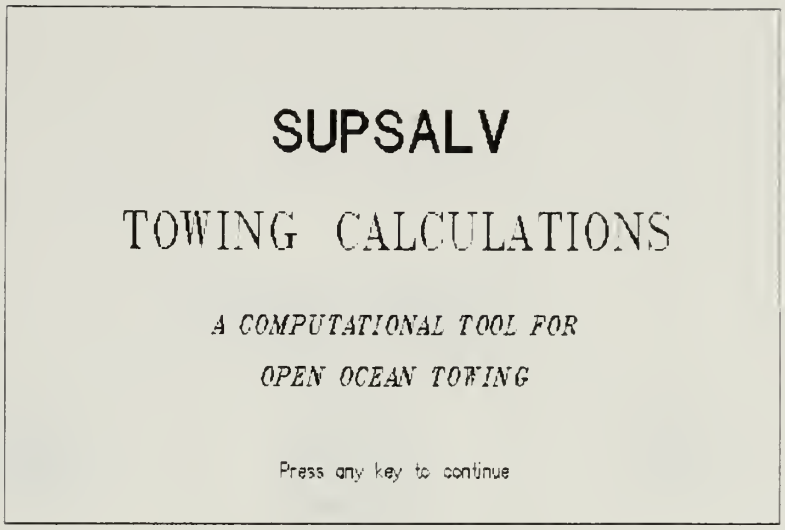

Figure 4.1. **TOWCALC** Title Screen

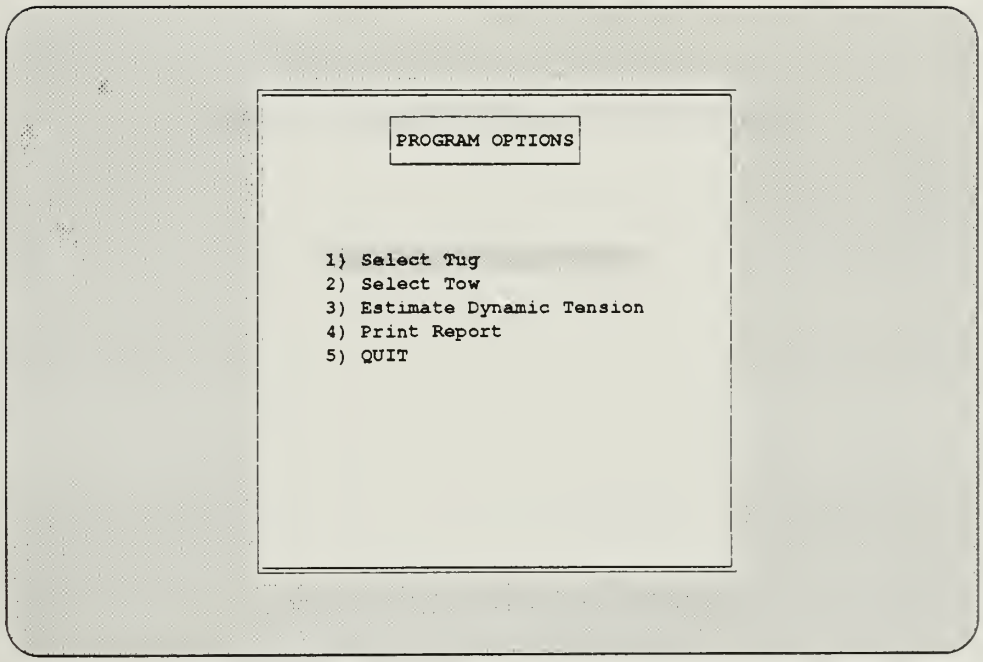

Figure 4.2. Main Program Options

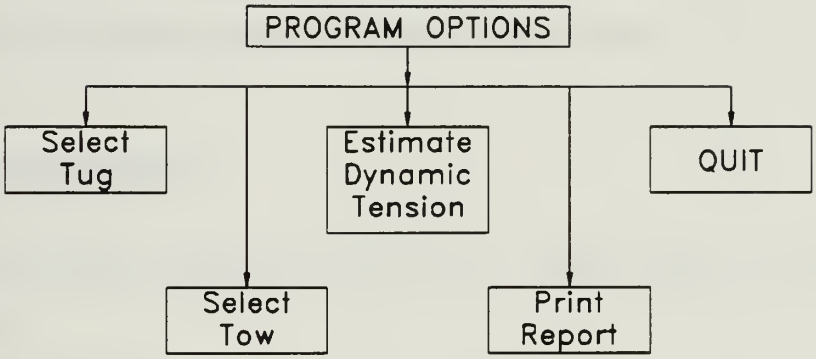

Figure 4.3. Program Options Menu Structure

A session of **TOWCALC** generally proceeds sequentially from **Select Tug** through **Print Report**. At startup, the only valid options are **Select Tug**, and **QUIT**; choosing any other option will generate an error message, requesting the user to select a tug. Once a tug is selected, **Select Tug**, **Select Tow**, and **QUIT** are valid options; choosing **Estimate Dynamic Tension** or **Print Report** prompt the user to select a tow. **Print Report** may only be selected once the previous options have been completed.

The following sections discuss each option in more detail.

4.2 Tug Selection

When this option is chosen, a second menu is displayed (Figure 4.4), giving the user three choices:

1) Enter new data

2) Edit existing data

3) Retrieve data from file

At the beginning of a session, only options 1) and 3) are allowed; choosing 2) generates an error message. Choosing option 3) will also produce an error if no tug data files have been previously saved.

Enter new data. Here, the user selects one of four Navy tug classes, chooses the tow-line length, and sets the size and length of the chain pendant. A popup menu is used to select the tug class (Figure 4.5), and the remaining data is typed directly (Figure 4.6).

The four tug classes are the ARS 38 class salvage ship, the ARS 50 class salvage ship, the ATS 1 class salvage ship, and the T-ATF 166 class fleet tug. Table 3.1 lists some of the principal characteristics of each ship class.

Once a tug class is chosen, **TOWCALC** automatically assigns the hawser diameter, as this is a characteristic of each class (see Table 4.1).

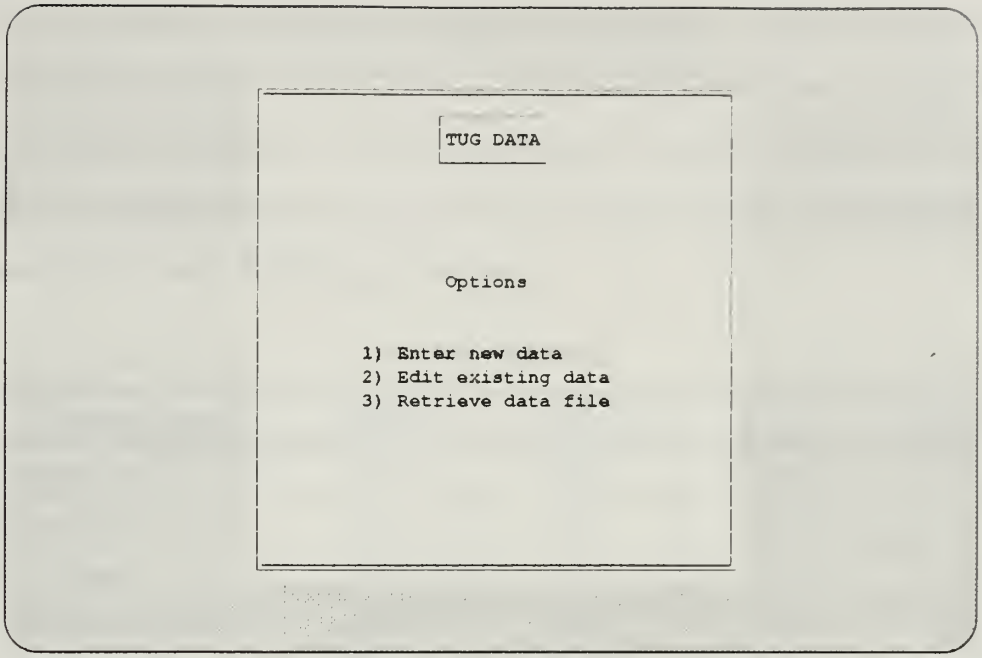

Figure 4.4. Tug Options

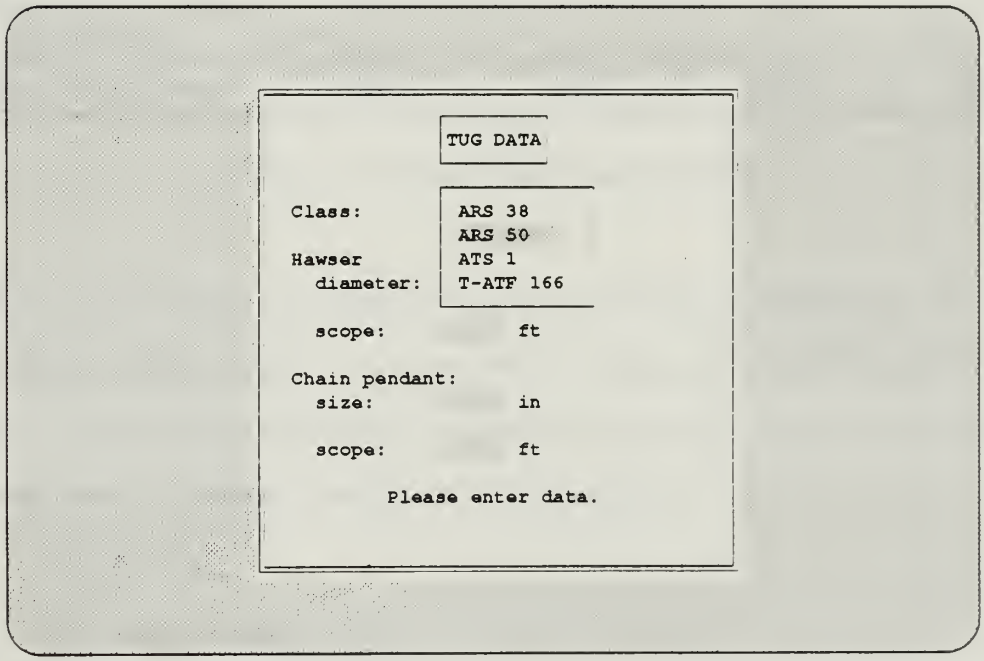

Figure 4.5. Selecting Tug Class

As data is entered, it is checked for errors and consistency. Invalid entries generate an error message prompting the user to try again. Negative lengths and sizes are disallowed. The towline length is compared to the maximum length of hawser available on the chosen tug—lengths exceeding this value are not allowed. Upper bounds on chain pendant size and length are arbitrarily set at 20 inches and 500 feet.

Characteristics	ARS 38	ARS 50	ATS 1	T-ATF 166
Length (feet)	213.5	255.0	282.7	225.0
Beam (feet)	43.0	52.0	50.0	42.0
Draft (feet)	16.0	17.5	18.0	15.0
Displacement (LT)	1900	3282	3117	2260
Cruising Range (nm / kts)	9400/12.5	8000/8.0	10000/13.0	10000/13.0
Maximum Sustained Speed (kts)	14.5	15.0	16.0	15.0
Shaft Horsepower	3000	4200	6000	7200

Table 4.1. Towing Vessel Characteristics

After the data is entered, the user has the option to edit his or her choices. The program asks if all data is correct: typing "yes" or pressing ↵ confirms this is so (Figure 4.6). If not, typing "no" prints a menu at the bottom of the display (Figure 4.7), showing the function key that must be pressed to change a given item. The choices are:

F1 Tug class

F2 Hawser length

F3 Chain pendant size

F4 Chain pendant length

Again, hawser size is set automatically. Pressing the "insert" key (Ins) quits editing.

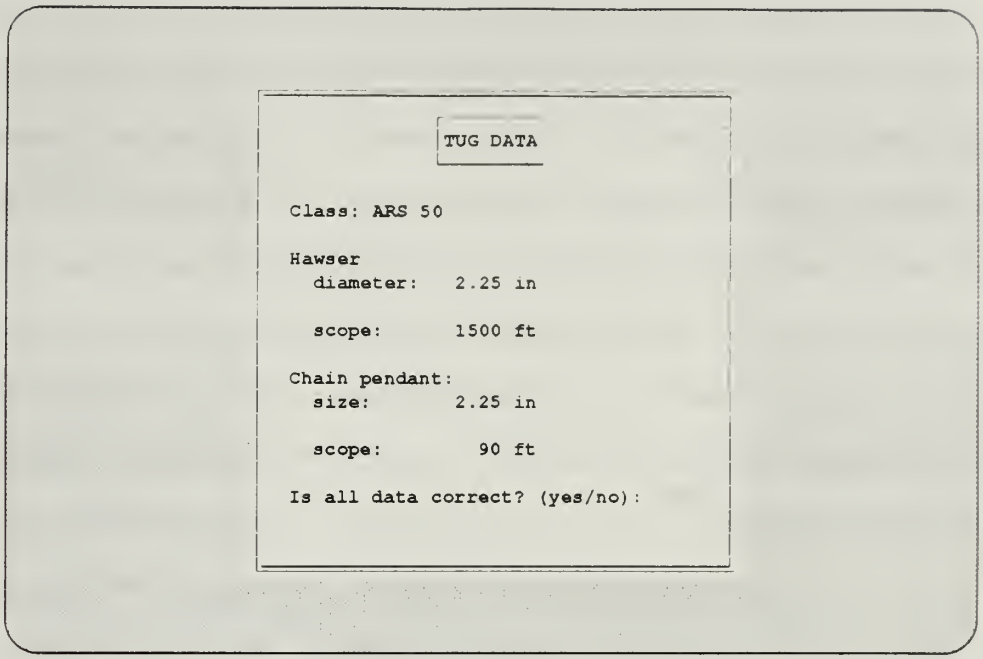

Figure 4.6. Entering Tug Data

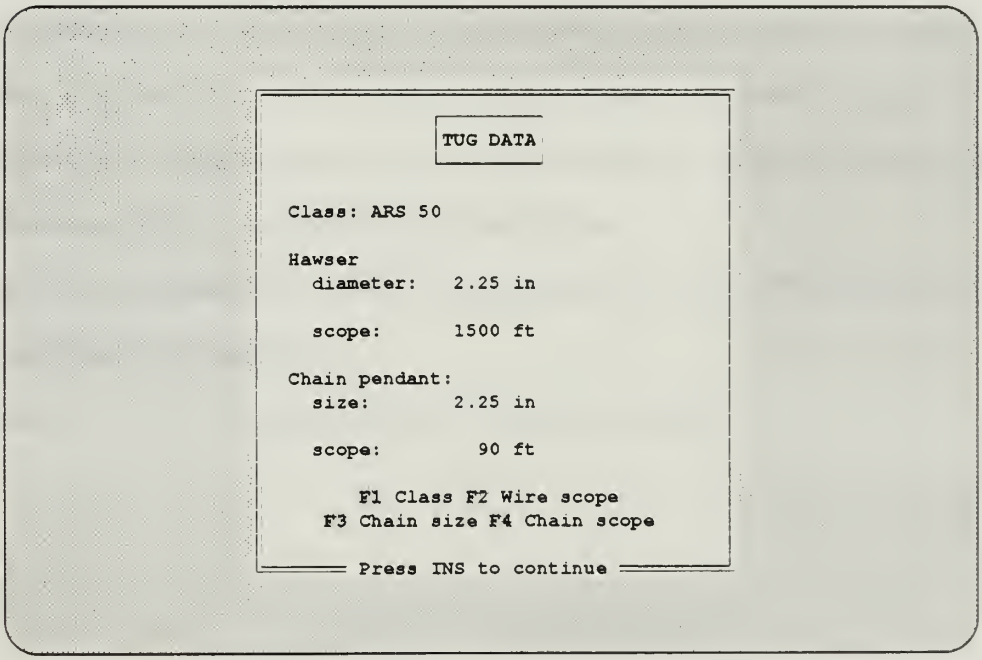

Figure 4.7. Editing Tug Data

Once all data is correct, the user is given the option to save the data to a file (Figure 4.8). Typing "yes" or pressing ↵ generates a prompt for a file name (eight characters maximum), and the data is then stored in a file on the computer's hard disk with the file extension ".TUG" (Figure 4.9). An error message is displayed if invalid characters are included in the file name (characters not allowed by DOS), or if the file could not be opened for some reason. The program then returns to the main menu, with **Select Tow** highlighted.

Edit existing data. Once a set of tug data have been entered during a given session, this option may be used to modify that data. The tug data summary is immediately displayed (Figure 4.6), with the prompt to confirm that data is correct. As before, typing "no" allows editing; typing "yes" or pressing ↵ displays the prompt to save to file.

Retrieve data from file. Choosing this option prompts the user to enter a tug file name (Figure 4.10). If a tug data file of this name has been saved on the hard disk, the program will retrieve the data stored there. If the file does not exist or can't be opened for some other reason, or if the file is not a tug file, an error message is displayed. Typing "Q" at the file name prompt will return the user to the tug options menu. Once data is retrieved, this option behaves exactly as the **Edit Existing data** option

In all cases, completion of **Select Tug** returns the user to the **PROGRAM OPTIONS** menu, with **Select Tow** highlighted.

Figure 4.11 shows the organization of this program module.

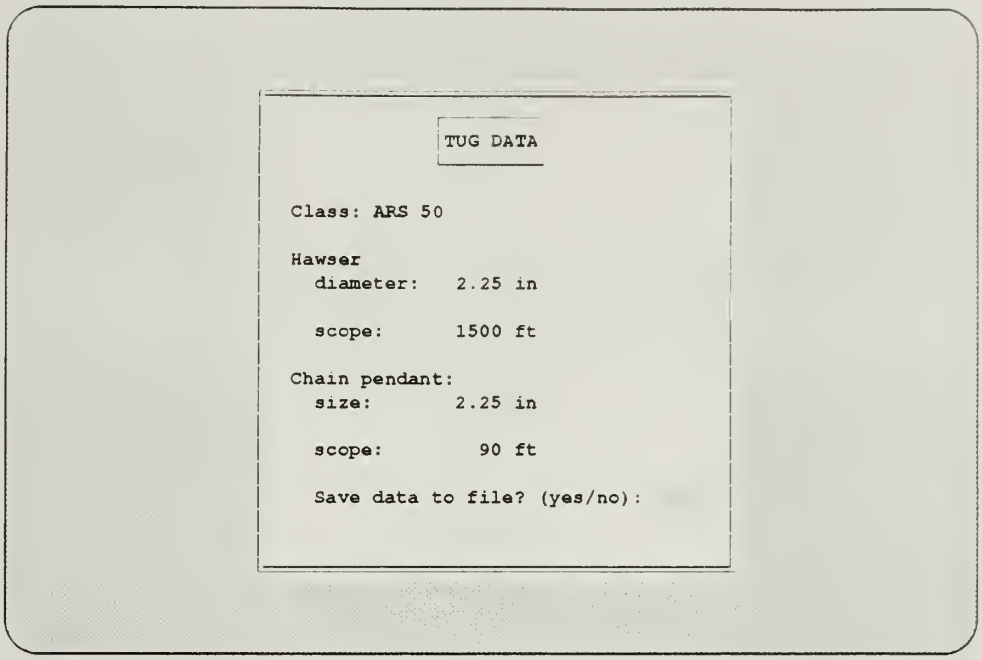

Figure 4.8. Save Tug File Prompt

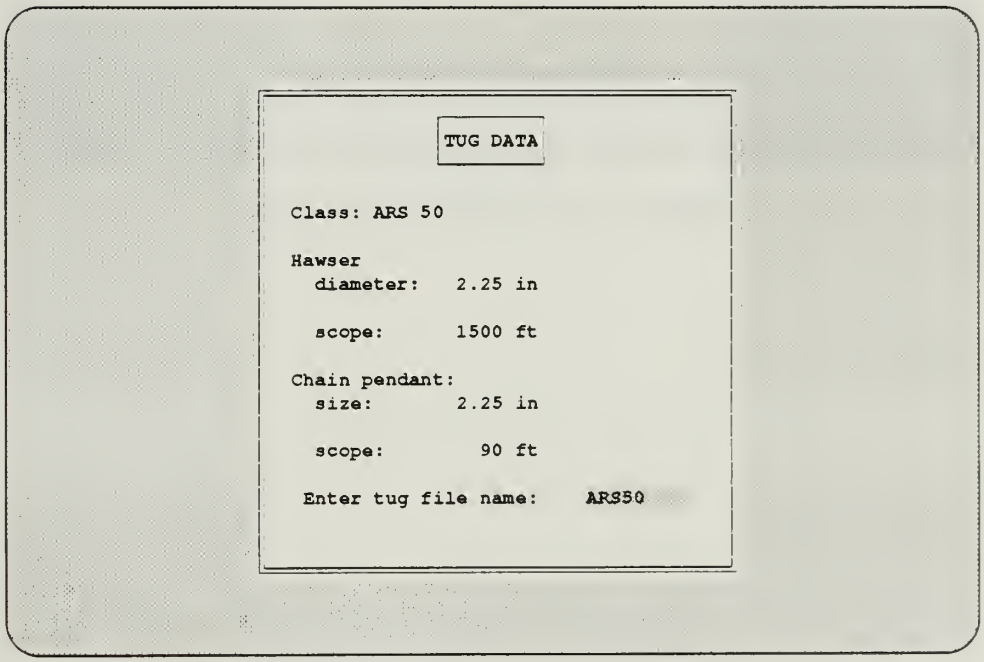

Figure 4.9. Entering Tug File Name

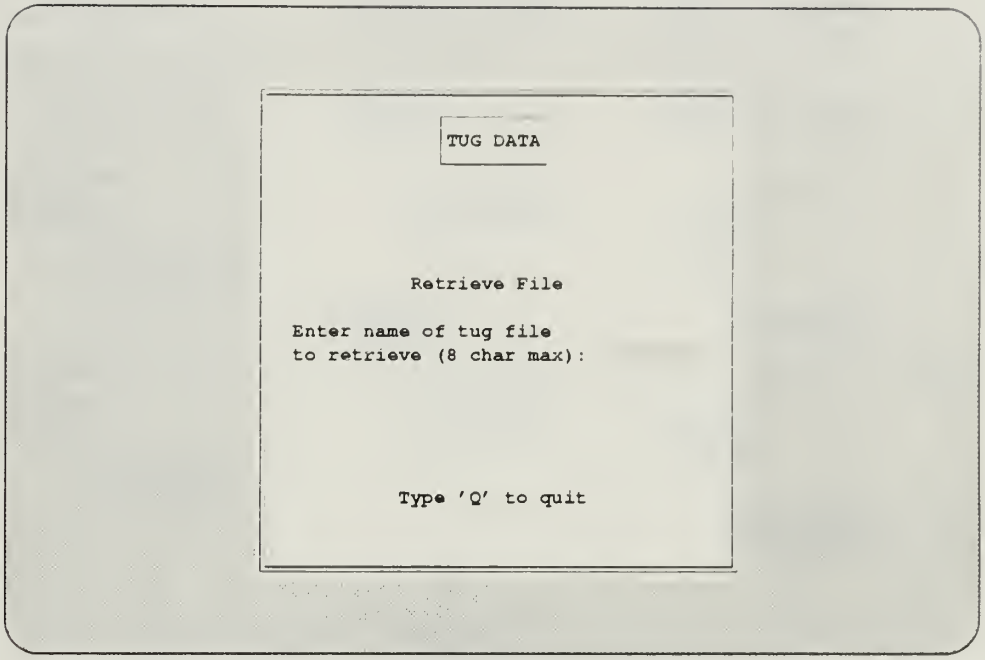

Figure 4.10. Retrieve Tug Data File

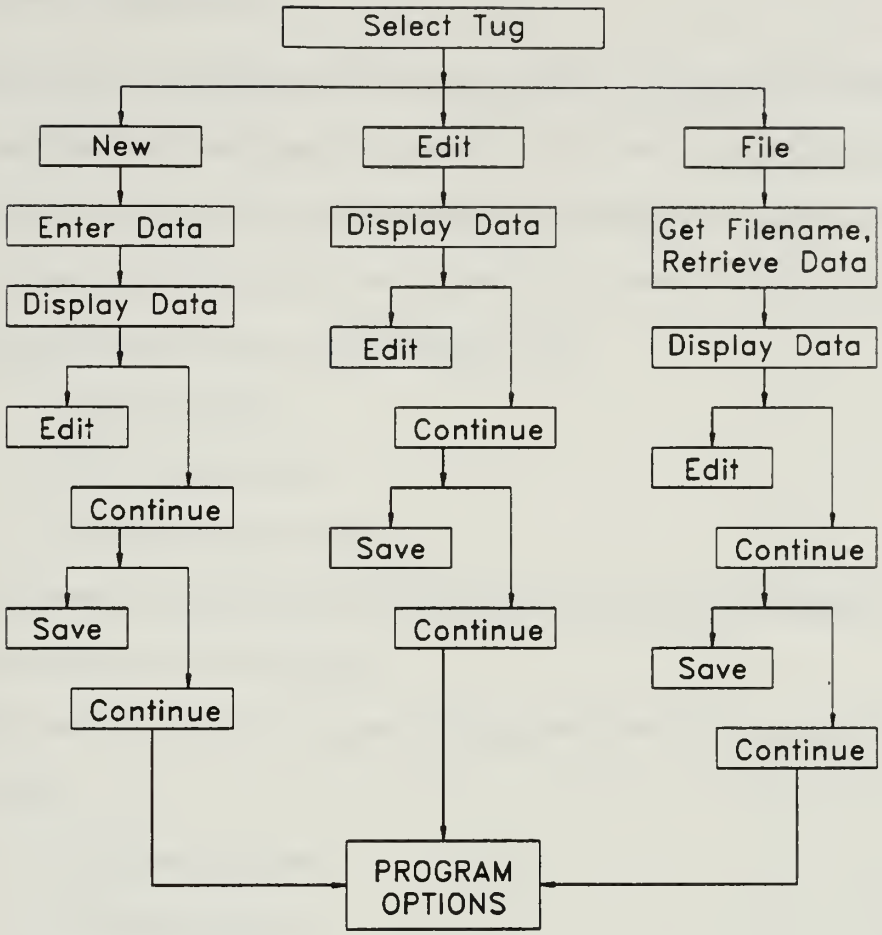

Figure 4.11. Program Flow: Select Tug Module

4.3 Tow Selection

This option accomplishes three separate functions: entering data, estimating tow and hawser resistance, and estimating tug capability. In this section we will look at each of the three functions in turn, focusing on the differences dictated by the type of tow vessel chosen.

As in the **Select Tug** option, when **Select Tow** is chosen a second menu is displayed, giving the same three choices as before (Figure 4.12):

1) Enter new data

2) Edit existing data

3) Retrieve data from file

The same comments regarding error messages apply here, except that option 3) generates an error if no tow files have been saved.

With each of these options, a menu is displayed to select the type of vessel to be towed (Figure 4.13):

1) Self-propelled ship

2) Floating drydock

3) Barge

These choices correspond to the three methods for estimating towing resistance discussed in Chapter 3.

4.3.1 Entering New Data

The process of entering data is similar for all three types of vessel, but the details differ because the data required to estimate resistance is different for each. Figure 4.14 shows this process schematically for self-propelled ships.

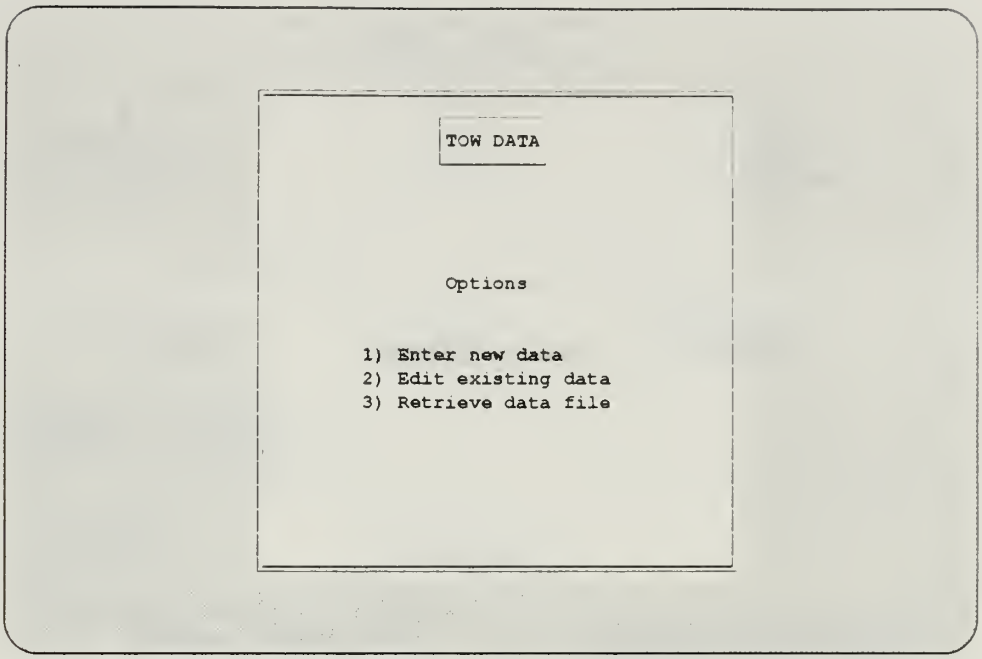

Figure 4.12. Tow Data Options

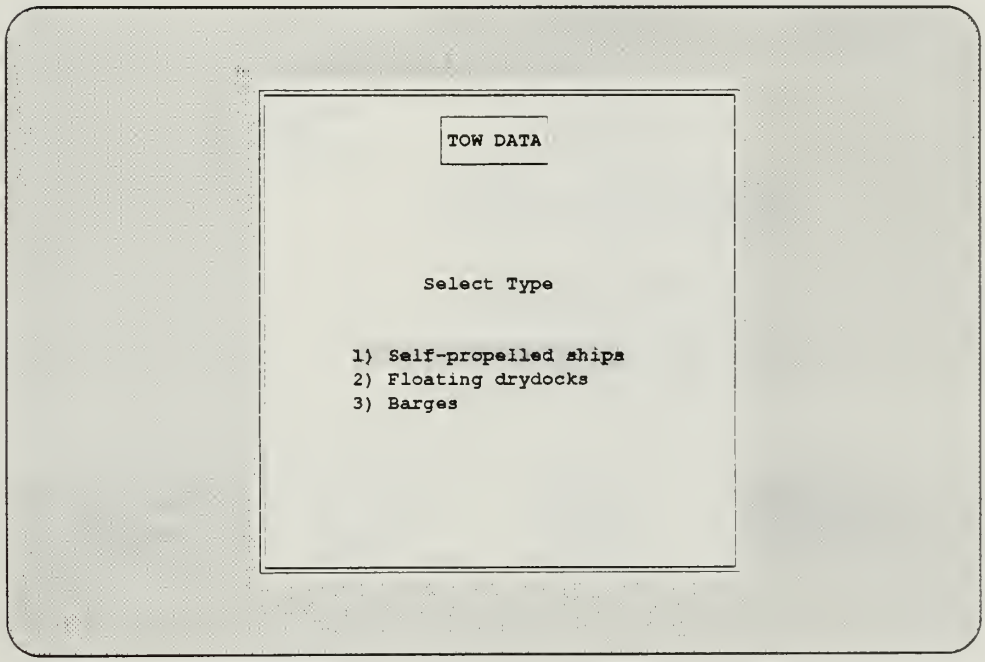

Figure 4.13. Select Tow Ship Type

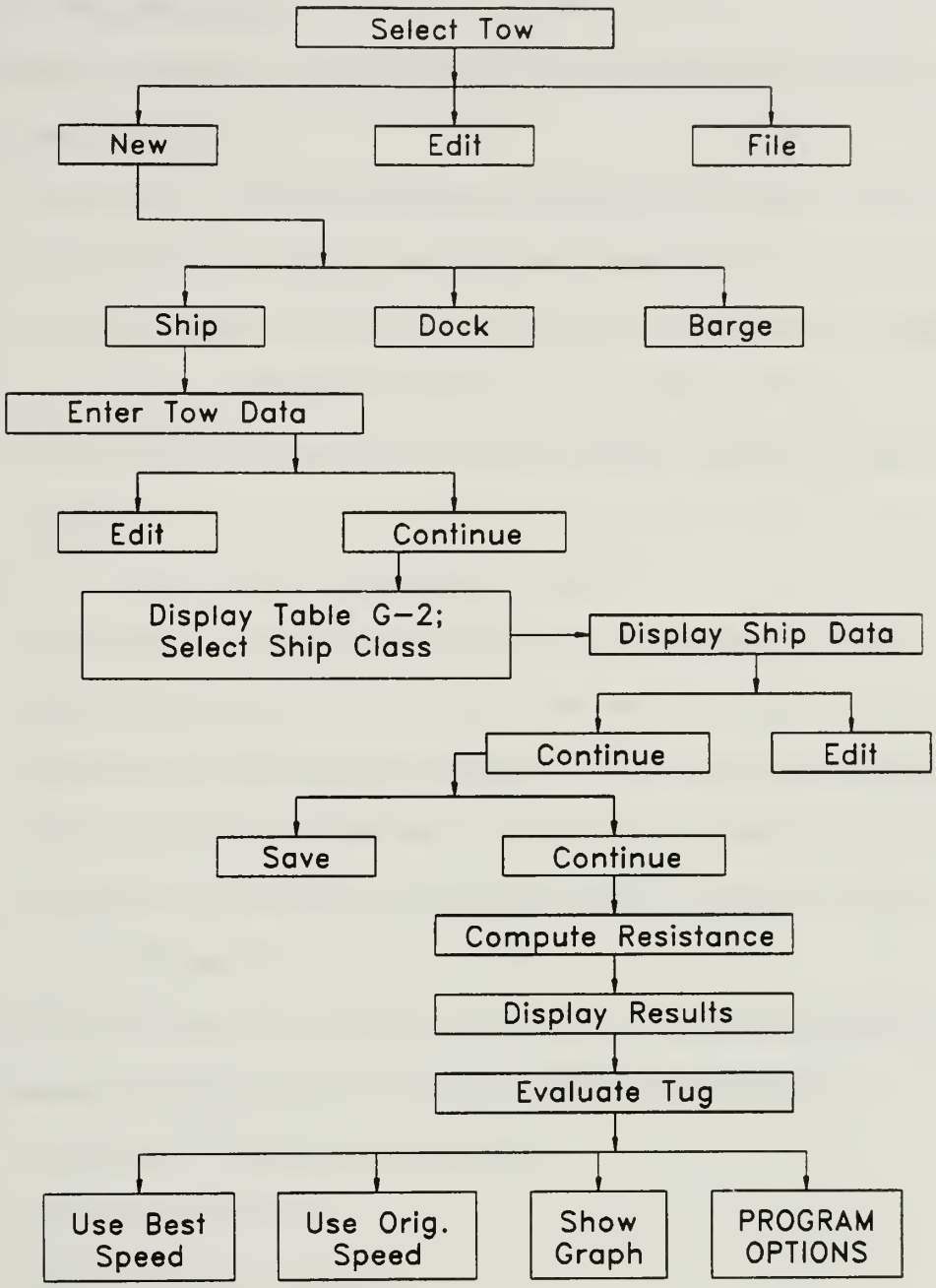

Figure 4.14. Program Flow: Enter New Data for Ships

4.3.1.1 Self-Propelled Ships

Entering Tow Data. When this ship type is chosen, the user is prompted to enter the following data (Figure 4.15):

- Hull number. This is the standard Naval designation for ships. For example, BB 63 refers to a battleship with hull number 63; CVN 65 refers to a nuclear-powered aircraft carrier with hull number 65. Entry may be made in either upper or lower case characters; **TOWCALC** converts all to upper case.
- Full load displacement. This is the ship's weight, expressed in long tons of 2240 pounds.
- Tow speed. The desired towing speed in knots.
- Wind speed. The maximum expected sustained wind speed in knots.
- Relative wind direction. The direction from which the wind is blowing with respect to the ship's heading, in degrees. Wind blowing from dead ahead has a relative direction of zero degrees; wind blowing from dead astern, 180 degrees.
- Propeller status. A popup menu gives the choices "Locked", "Trailing", or "Removed" (Figure 4.16).

As in **Select Tug**, data is checked for errors and consistency. Invalid entries generate an error message prompting the user to try again. Errors will be given for:

- numeral as first character in hull number,
- negative displacement,
- negative tow or wind speed,
- displacement greater than 100,000 long tons,
- tow speed greater than 12.0 knots,
- wind speed greater than 48.0 knots,
- relative wind direction greater than 360 degrees, or between 70 and 110 degrees.

The limit on displacement is enforced because the data base of ship types has no information for larger ships. Similarly, 48.0 knots wind speed is the upper limit for the added resistance curves. The extreme tension data base does not cover beam seas, associated with wind directions between 70 and 110 degrees relative, primarily because the motions of the tug under such conditions, particularly roll, are more restrictive than towline tensions. Towing speed is limited to 12.0 knots partly due to safety considerations, and partly due to the power limitations of Navy towing vessels.

Warning messages will be given for the following conditions:

- displacement less than 350 long tons, or greater than 91,000 long tons but less than or equal to 100,000 long tons,
- tow speed greater than 10.0 knots but less than or equal to 12.0 knots,
- relative wind direction less than zero or between 180 and 360 degrees.

Displacements in the warning range are not covered specifically by the Towing Manual data base, but reasonable estimates may be made using data for the ship listed that is nearest in size. Tow speeds greater than 10.0 knots are unusual, but may occur under special circumstances, so **TOWCALC** accepts speeds in this range. Relative wind directions in the range given here are converted to the corresponding angle between zero and 180 degrees.

Editing. Once the data is entered, **TOWCALC** prompts the user to verify that all is correct. As before, if all data is correct, typing "yes" or pressing ↵ confirms that this is so. If not, typing "no" prints a menu at the bottom of the display, with the choices (Figure 4.17):

F1 Hull number

F2 Full load displacement

F3 Tow speed

F4 Maximum expected wind speed

F5 Relative wind direction

F6 Propeller status

To quit editing, press Ins .

Selecting Ship Class. In order to make resistance predictions using the method of Chapter 3, **TOWCALC** must make use of the data provided by Table G-2 of the Towing Manual. This data is stored in a file on the computer, but the user must specify which ship class to use. **TOWCALC** displays a portion of this data file on the screen, and the cursor keys (↑, ↓, ←, →, PgUp, PgDn, Home, End) are used to move the highlighted bar from choice to choice. Pressing ↵ makes the selection. Initially, **TOWCALC** searches the data file for the closest match to the hull number the user entered, and displays that ship type at the top of the screen (Figure 4.18). If it cannot find an exact match, it looks for a matching hull type, such as "DDG" or "CVN", and displays the first ship of that type at the top of the screen. If no match is found, the first ship type in the list is displayed.

Along with the hull number of the ship class, **TOWCALC** displays a description, such as "GUIDED MISSILE DESTROYER", the displacement in long tons, the frontal projected windage area, and the projected propeller area. These are to help the user make the best selection possible.

After the user selects a ship class, **TOWCALC** displays the following data and prompts the user for verification:

- hull number entered by the user,
- ship class selected,
- actual displacement,
- tabulated displacement,
- frontal projected windage area,
- wind coefficient,
- propeller area,

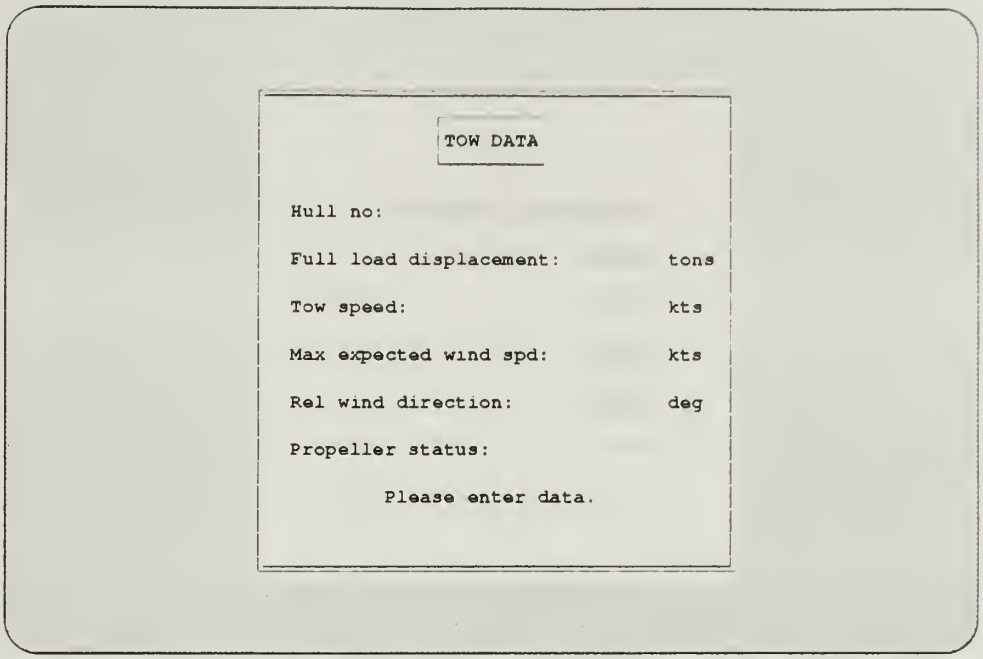

Figure 4.15. Entering Tow Data for Self-Propelled Ships

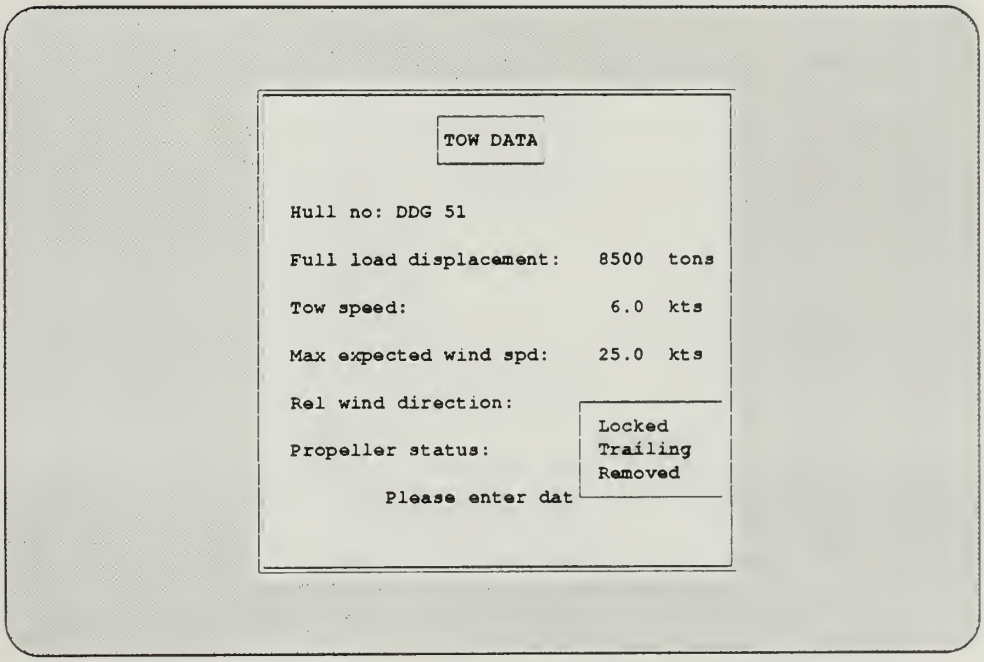

Figure 4.16. Propeller Status Menu

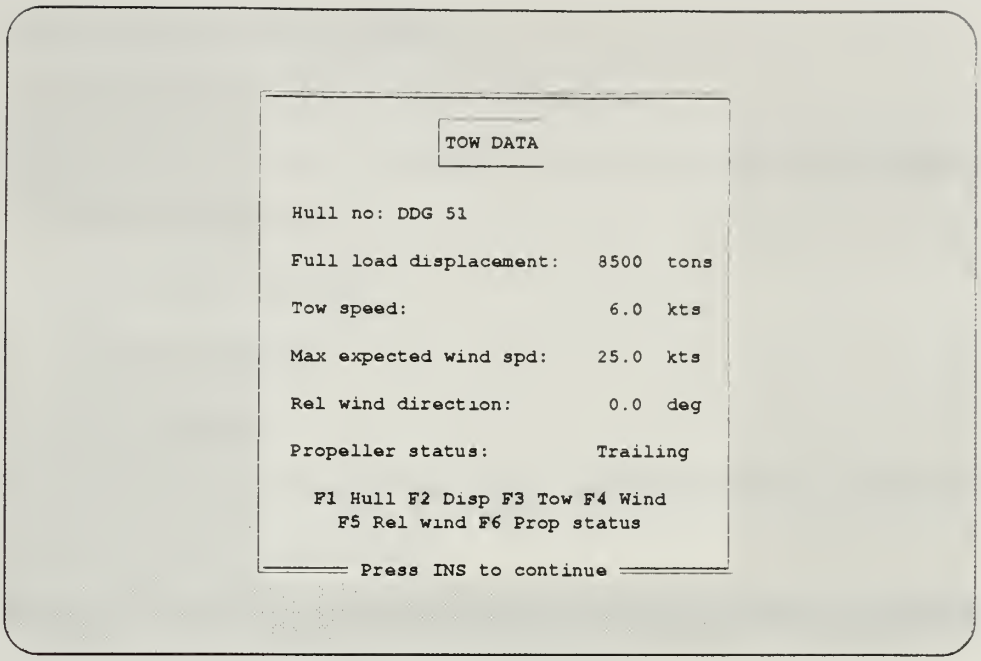
TOW DATA
Hull no: DDG 51
Full load displacement: 8500 tons
Tow speed: 6.0 kts
Max expected wind spd: 25.0 kts
Rel wind direction: 0.0 deg
Propeller status: Trailing
F1 Hull F2 Disp F3 Tow F4 Wind
F5 Rel wind F6 Prop status
Press INS to continue

Figure 4.17. Editing Tow Data for Ships

CLASS	DESCRIPTION	DISP	WINDAGE AREA	PROP AREA
DDG 51-53	GUIDED MISSILE DESTROYERS	8,300	6,900	254
DDG 993-996	GUIDED MISSILE DESTROYERS	8,300	5,000e	254
DDG 37-46 (ex DLG 6/9)	GUIDED MISSILE DESTROYERS	6,150	3,000e	228
DDG 2-24	GUIDED MISSILE DESTROYERS	4,500	2,256	176
DDG 31-34	GUIDED MISSILE DESTROYERS	4,150	2,100	194
DD 963-992,997	DESTROYERS	7,810	4,400	254
DD 931-951	DESTROYERS	4,200	2,100	194
DD 445 CLASS	DESTROYERS	3,040	1,400	134
DD 692 CLASS	DESTROYERS	3,400	1,400	158
DD 710 CLASS	DESTROYERS	3,540	1,450	158
DE 1006 CLASS	DESTROYER ESCORT	1,914	1,342	79
FFG 7-61	GUIDED MISSILE FRIGATES	3,585	2,200	170e
FFG 1-6 (DEG)	GUIDED MISSILE FRIGATES	3,426	1,715	131
FF 1052-1097 (DE)	FRIGATES	3,900	2,020	131
FF 1040-1051 (DE)	FRIGATES	3,400	1,715	131
LCC 19-20	AMPHIBIOUS COMMAND SHIPS	18,650	7,360	220e
LHA 1-5	AMPHIBIOUS ASSAULT SHIPS	39,300	11,500	262
LPH 2-12	AMPHIBIOUS ASSAULT SHIPS	18,800	6,700	155
LPD 4-15	AMPHIBIOUS TRANSPORT DOCKS	17,000	8,350	175
LPD 1-2	AMPHIBIOUS TRANSPORT DOCKS	14,665	8,300	175

(Items marked with 'e' are best estimate.)

Use arrow keys, [pgup], [pgdn], [home], [end] to view choices.

Figure 4.18. Selecting Ship Class

- curve number for hull resistance,
- curve number for added resistance due to waves.

If data needs editing, an edit menu is displayed, as before, with the choices (Figure 4.19):

F1 Actual displacement

F2 Frontal windage area

F3 Wind coefficient

F4 Propeller area

The resistance curve numbers are not subject to editing, as these are unique to the ship class selected. Pressing Ins ends editing.

Saving to File. The steps to save tow data to a file are the same as in **Select Tug**, except data is stored in a file with the extension ".TOW".

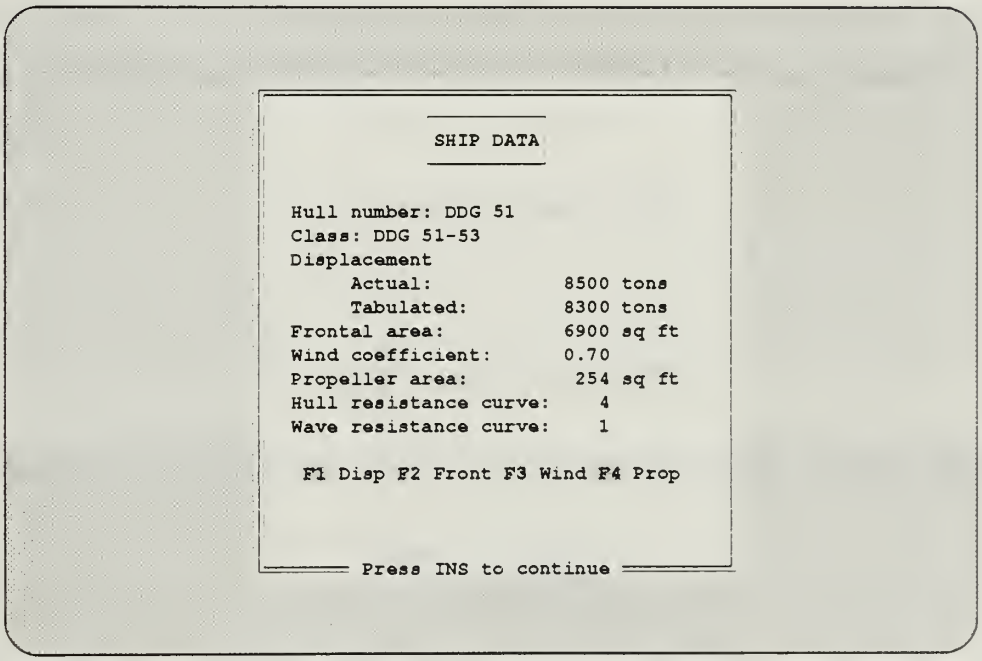

Figure 4.19. Editing Ship Data

4.3.1.2 Floating Drydocks

Entering Tow Data. When this ship type is chosen, a popup menu is displayed showing the fourteen drydock types listed in Table 3.1 (Figure 4.20). When the user makes a choice, **TOWCALC** reads the data for that drydock type from a file. Then the user is prompted to enter a value indicating the hull condition of the drydock: this is a scale from zero to ten, with zero representing a clean bottom and ten representing a bottom heavily fouled with marine growth (Figure 4.21). The scale value is used to compute the value of the coefficient f_1 in Table 3.1. Next, **TOWCALC** asks for the tow speed, wind speed, and relative wind direction (Figure 4.22).

Data is checked for errors and consistency, as always. Invalid entries generate error messages; the same criteria for ship tow data apply here.

Estimating Displacement. The Towing Manual does not provide the displacement of drydocks in Table 3.1. This information is needed later, though, when matching the drydock against the ships in the extreme tension data base. **TOWCALC** estimates the displacement, Δ, as follows

$$\Delta = C_b \times \text{Length} \times \text{Beam} \times \text{Draft}$$

where

$$C_b \equiv \frac{\Delta}{L \times B \times T} \equiv \text{block coefficient}$$

Drydock length and beam are found in Table 3.1, but the draft must be estimated using

$$\text{Draft} = \frac{B}{C_x \times \text{Beam}}$$

where

$$C_x \equiv \frac{B}{\text{Beam} \times \text{Draft}} \equiv \text{Maximum section coefficient}$$

and B is the underwater cross-sectional area, given in Table 3.1. The block coefficient, C_b, and the maximum section coefficient, C_x, are approximated as

$$C_b = 0.8$$

$$C_x = 0.9$$

Editing. Once the data is entered, **TOWCALC** displays a summary and prompts the user to verify that all is correct. If it is not, typing "no" prints a menu at the bottom of the display, with the choices (Figure 4.23):

F1 Drydock type

F2 Hull condition

F3 Tow speed

F4 Wind speed

F5 Relative wind direction

Pressing F1 displays the popup menu of drydocks as before; selecting a different drydock causes **TOWCALC** to read a new set of coefficients from file. Pressing Ins quits editing, as always.

Saving to File. The steps here are identical with those for self-propelled ships.

Figure 4.24 shows schematically the process for entering drydock data.

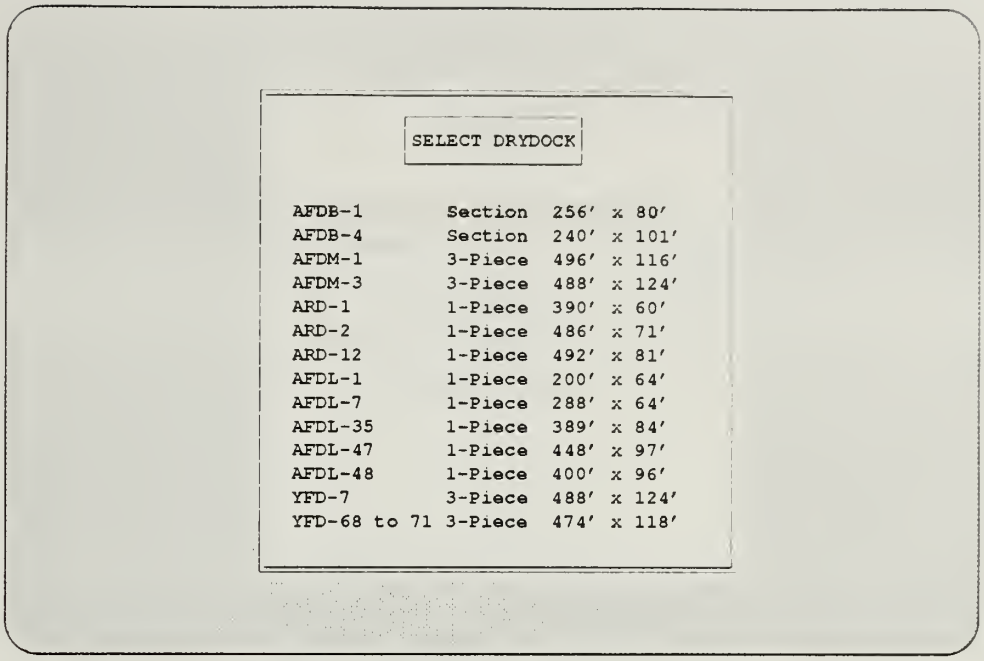
SELECT DRYDOCK
AFDB-1 Section 256' x 80'
AFDB-4 Section 240' x 101'
AFDM-1 3-Piece 496' x 116'
AFDM-3 3-Piece 488' x 124'
ARD-1 1-Piece 390' x 60'
ARD-2 1-Piece 486' x 71'
ARD-12 1-Piece 492' x 81'
AFDL-1 1-Piece 200' x 64'
AFDL-7 1-Piece 288' x 64'
AFDL-35 1-Piece 389' x 84'
AFDL-47 1-Piece 448' x 97'
AFDL-48 1-Piece 400' x 96'
YFD-7 3-Piece 488' x 124'
YFD-68 to 71 3-Piece 474' x 118'

Figure 4.20. Select Drydock Class

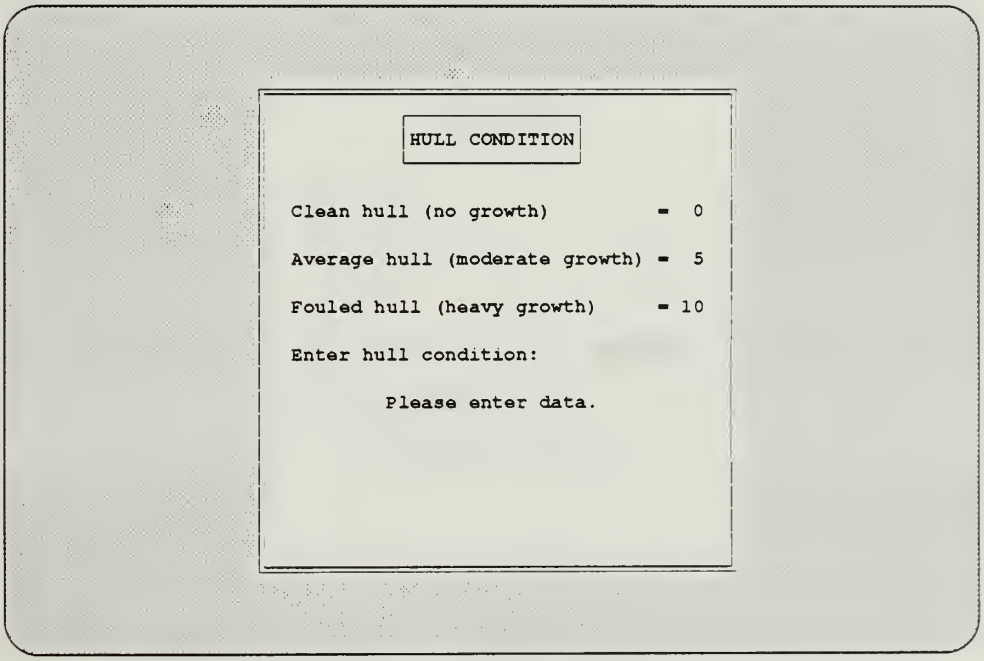
HULL CONDITION
Clean hull (no growth) = 0
Average hull (moderate growth) = 5
Fouled hull (heavy growth) = 10
Enter hull condition:
Please enter data.

Figure 4.21. Entering Hull Condition

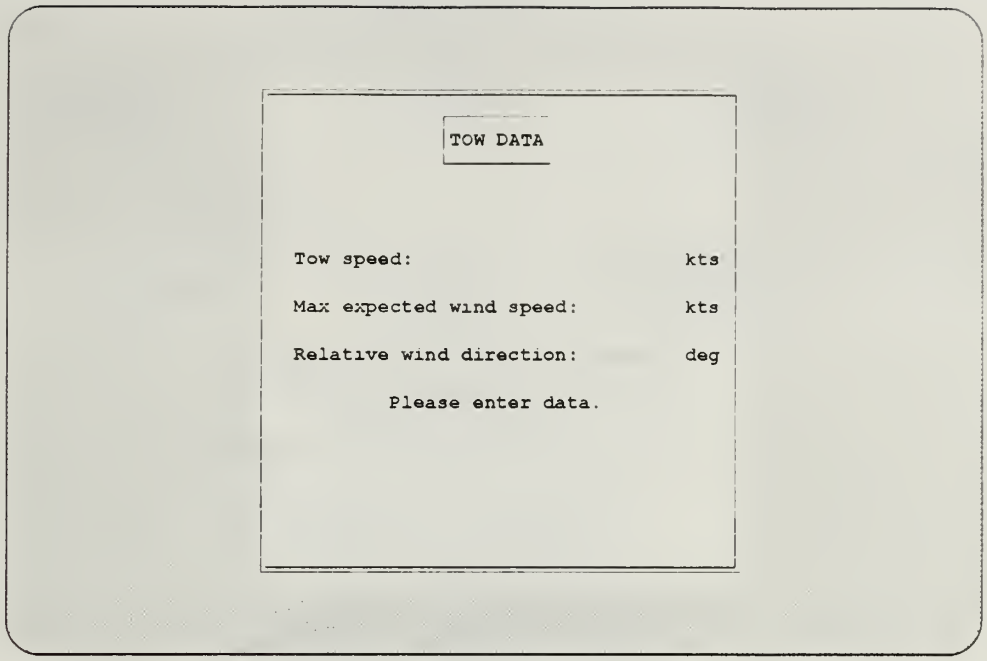

Figure 4.22. Entering Drydock Tow Data

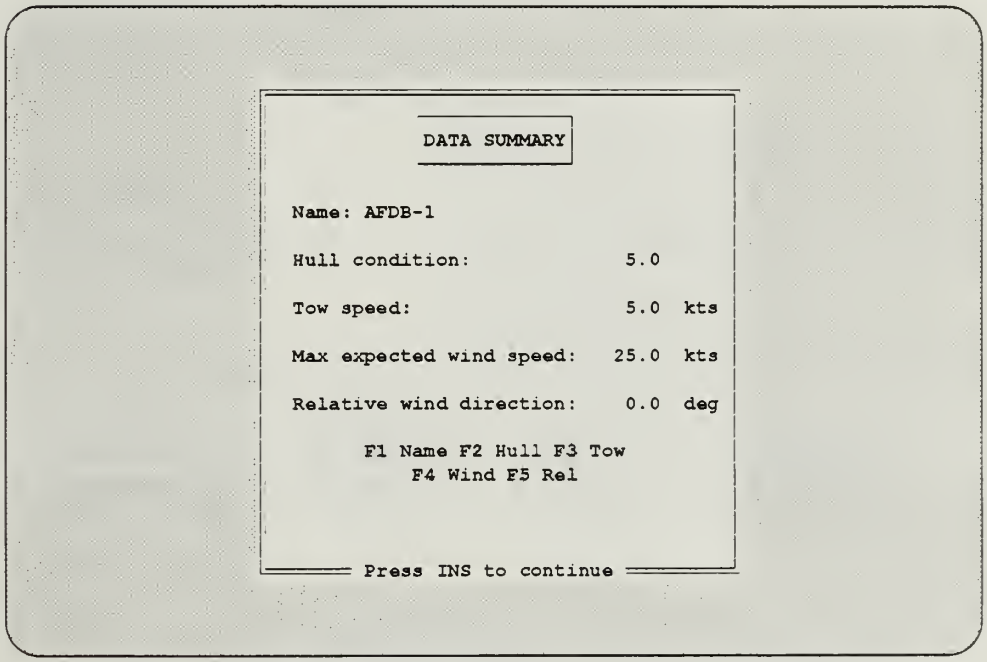
DATA SUMMARY
Name: AFDB-1
Hull condition: 5.0
Tow speed: 5.0 kts
Max expected wind speed: 25.0 kts
Relative wind direction: 0.0 deg
F1 Name F2 Hull F3 Tow
F4 Wind F5 Rel
Press INS to continue

Figure 4.23. Editing Drydock Tow Data

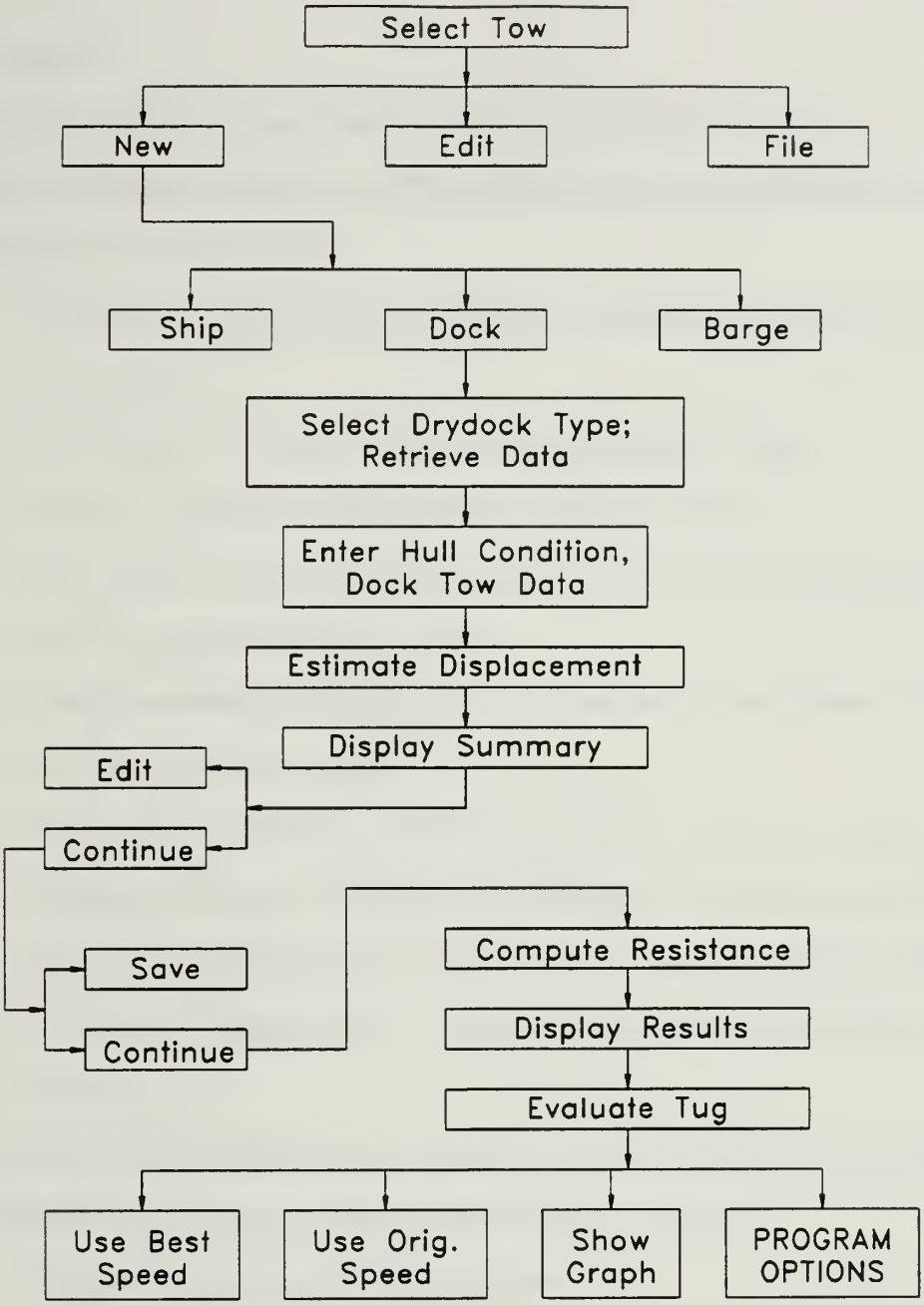

Figure 4.24. Program Flow: Enter New Data for Drydocks

4.3.1.3 Barges

Figure 4.25 shows schematically the process of entering barge data.

Entering Barge Dimensions. When this ship type is chosen, the user is prompted for the following data (Figure 4.26):

- Barge name. This is generally the Naval hull designation, such as "YFN 89" or "YC 1170".
- Hull length. The length of the barge at the waterline, in feet.
- Beam. The beam of the barge at the waterline, in feet.
- Hull depth. The depth of the barge from the keel to the main deck, in feet.
- Draft. The draft of the barge, in feet.
- Deckhouse length. The length, in feet, of the barge's deckhouse, if one is present, or of any deck cargo, if not.
- Deckhouse width. The width of the deckhouse or deck cargo, in feet.
- Deckhouse height. The height of the deckhouse or deck cargo, in feet.
- End shape. A popup menu is displayed giving the choices "Rake ended", "Ship ended", and "Square ended" to indicate the shape of the bow and stern of the barge (Figure 4.27).

As always, data is checked for errors and consistency. Invalid entries generate an error message prompting the user to try again. Errors will be given for:

- numeral as first character in hull number,
- negative lengths,
- hull length greater than 1000 feet,
- beam greater than 500 feet,
- hull depth greater than 100 feet,
- draft deeper than hull depth,
- deckhouse length greater than hull length,

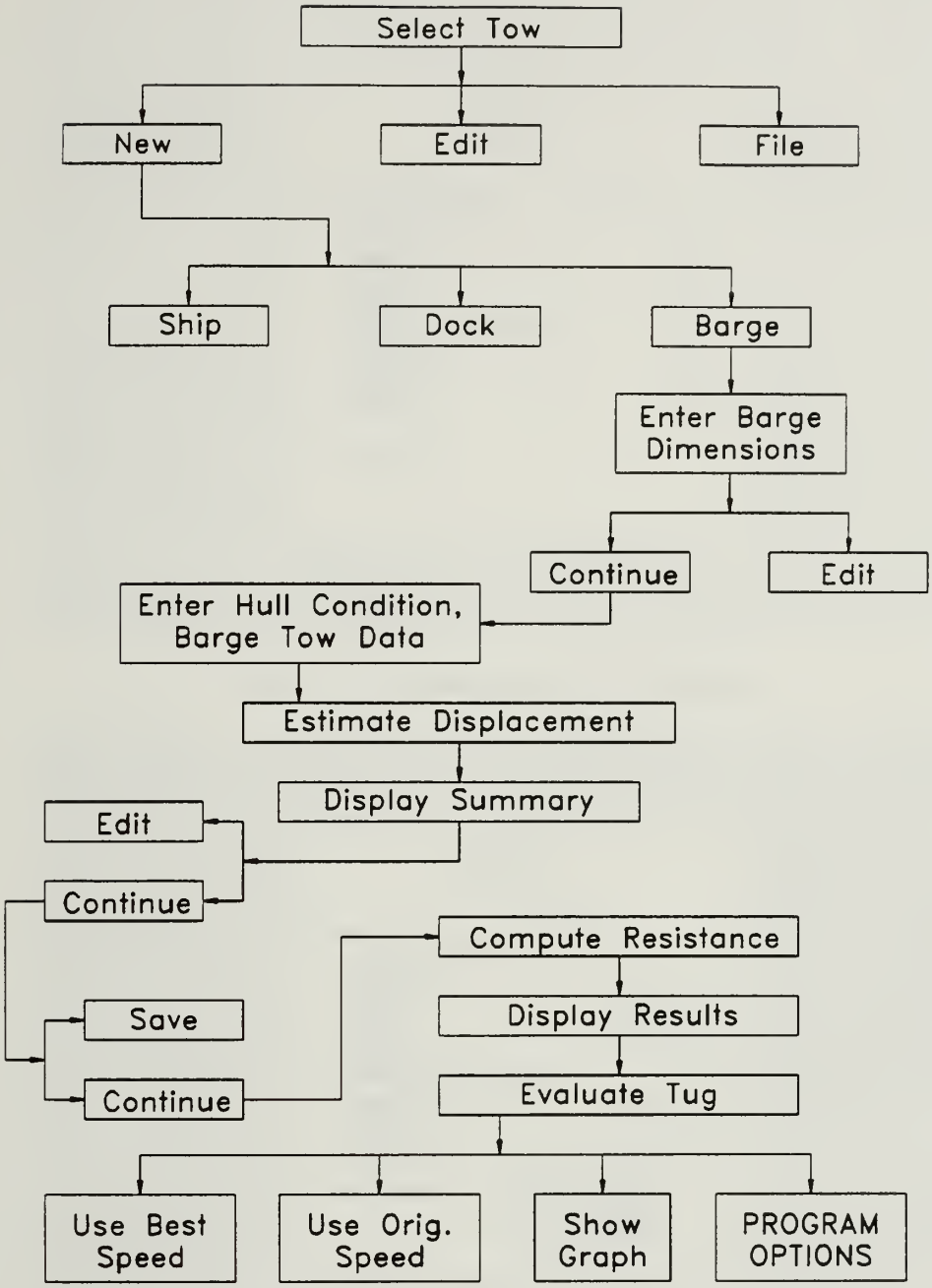

Figure 4.25. Program Flow: Entering New Data for Barges

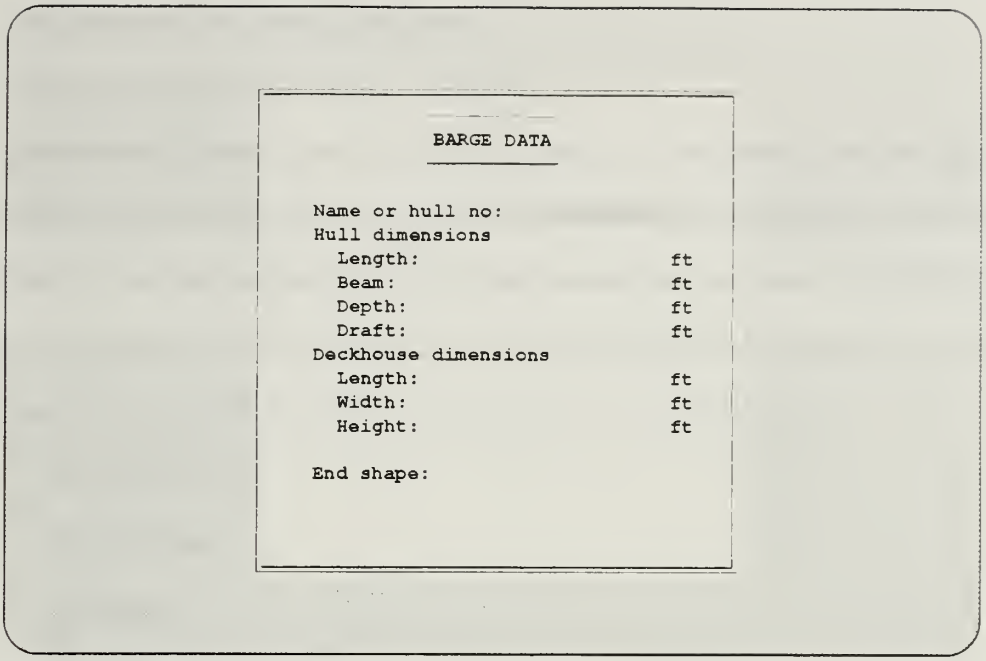

Figure 4.26. Entering Barge Dimensions

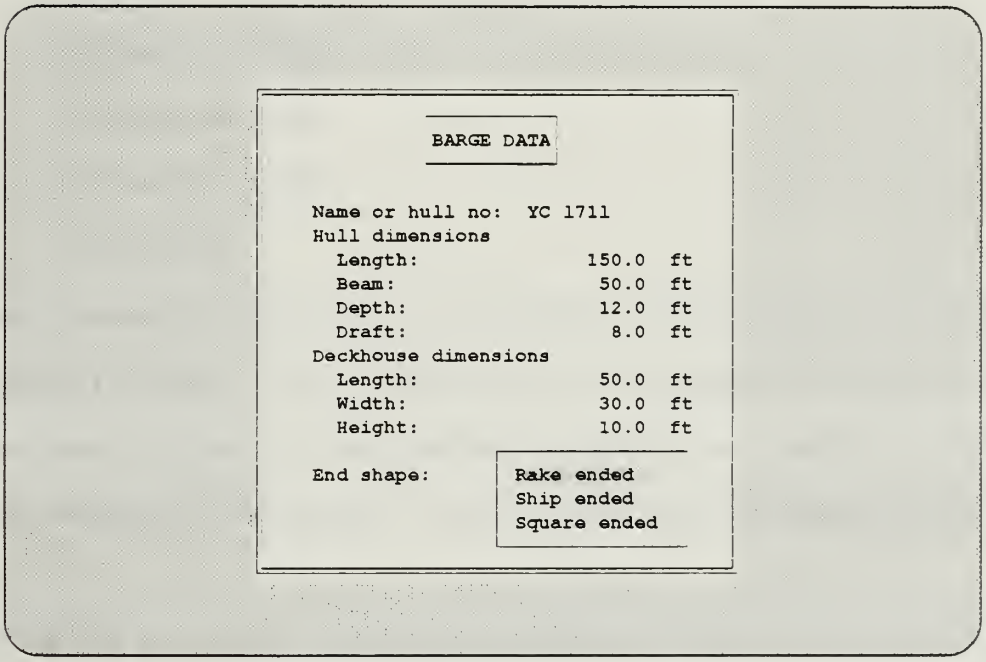

Figure 4.27. Barge End Shape Menu

- deckhouse width greater than beam,
- deckhouse height greater than 100 feet.

The upper bounds on dimensions are somewhat arbitrary, as barges are rarely as large as the maximum dimensions specified here. The intent is to catch typing mistakes by the user.

Editing Barge Dimensions. Once the barge dimensions are entered, **TOWCALC** prompts the user for verification; typing "no" at the prompt prints the editing function key choices along the left side of the display, corresponding to (Figure 4.28):

F1 Barge name

F2 Hull length

F3 Beam

F4 Hull depth

F5 Draft

F6 Deckhouse length

F7 Deckhouse width

F8 Deckhouse height

F9 End shape

Any changes made are checked for errors and consistency. Pressing Ins quits editing.

Entering Tow Data. After the dimensions have been entered and confirmed, **TOWCALC** prompts for the same tow data requested for drydocks: hull condition, tow speed, wind speed, and relative wind direction. The same error checks are made here as for drydock tow data.

Editing Tow Data. Once all data have been entered, the usual prompt appears for editing. The procedure is the same—the menu is printed at the bottom of the display, with the choices (Figure 4.29):

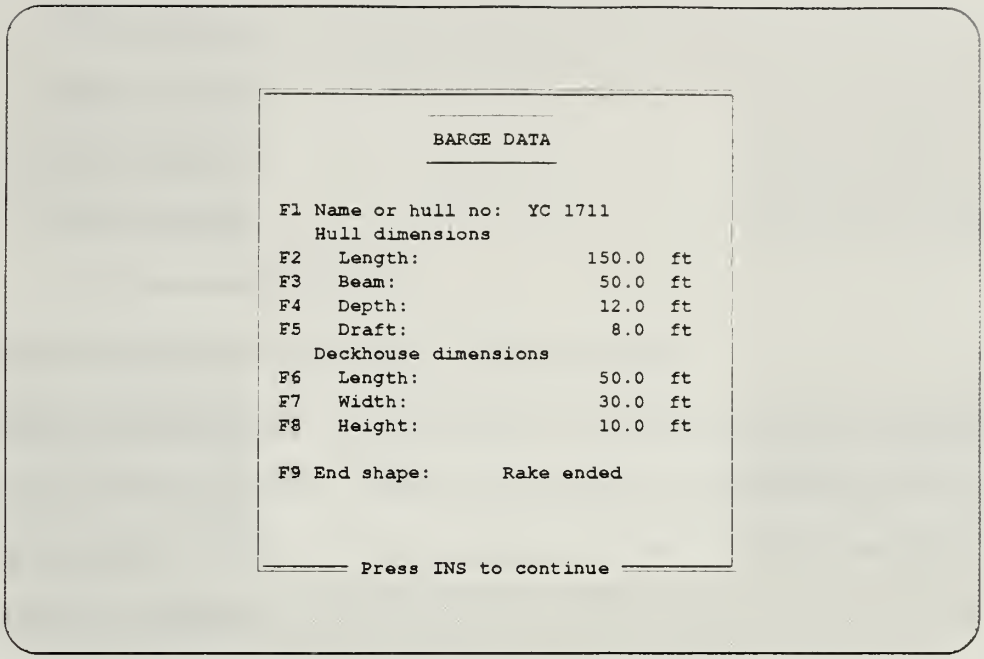

Figure 4.28. Editing Barge Dimensions

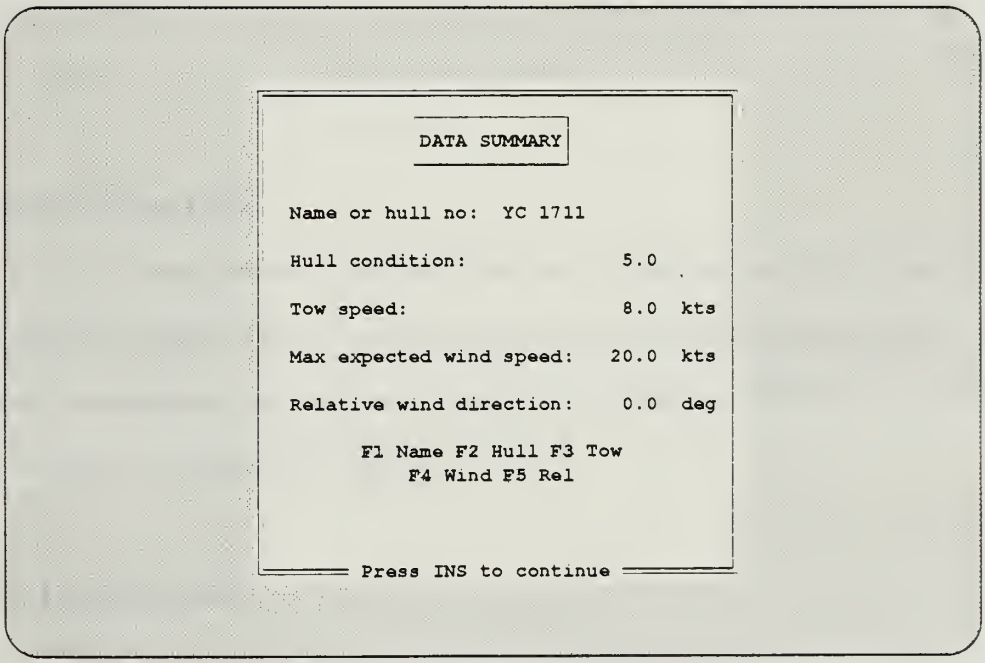
DATA SUMMARY
Name or hull no: YC 1711
Hull condition: 5.0
Tow speed: 8.0 kts
Max expected wind speed: 20.0 kts
Relative wind direction: 0.0 deg
F1 Name F2 Hull F3 Tow
F4 Wind F5 Rel
Press INS to continue

Figure 4.29. Editing Barge Tow Data

F1 Barge name

F2 Hull condition

F3 Tow speed

F4 Wind speed

F5 Relative wind direction

Any changes are checked for errors; pressing Ins quits editing.

Estimating Displacement. The displacement of the barge is needed for comparison with the extreme tension data base. **TOWCALC** estimates the displacement of barges using the same relationship as for drydocks, and using the same values of block coefficient and maximum section coefficient:

$$\Delta = C_b \times \text{Length} \times \text{Beam} \times \text{Draft}$$

Saving to File. The steps here are identical with those for self-propelled ships and floating drydocks.

4.3.2 Editing Existing Data

Once data has been entered for a particular tow, the user may make changes by choosing the **Select Tow** option from the main menu, then choosing **Edit existing data**. The process is shown schematically in Figure 4.30, and since the steps are different for each ship type, they will each be discussed in turn.

4.3.2.1 Self-Propelled Ships

Here, **TOWCALC** immediately displays the tow data entered previously, in the same format. The edit prompt appears, and the menu of choices is the same as before (Figure 4.17).

Next, **TOWCALC** displays the ship data, and prompts for editing (Figure 4.19). Note there is no option to change the ship class: to do so, the user must return to the **Select Tow** option and follow the original path for entering new data for a self-propelled ship.

Once editing is complete, the user is prompted to save the data to a file, as before.

4.3.2.2 Floating Drydocks

This option displays a summary of all drydock data, and prompts for editing. The edit menu is the same as before, and **TOWCALC** behaves the same as when entering new data. After editing, the user is prompted to save the current data to file.

4.3.2.3 Barges

TOWCALC displays a summary of barge dimensions and prompts for editing (Figure 4.28). After editing the dimensions, the barge tow data (hull condition, tow speed, etc.) are displayed and another editing prompt is given (Figure 4.29). Finally, the user is prompted to save data to file.

4.3.3 Retrieving Data Files

The procedures for retrieving tow data files are similar to those in **Select Tug**; Figure 4.31 shows the steps involved for all three ship types.

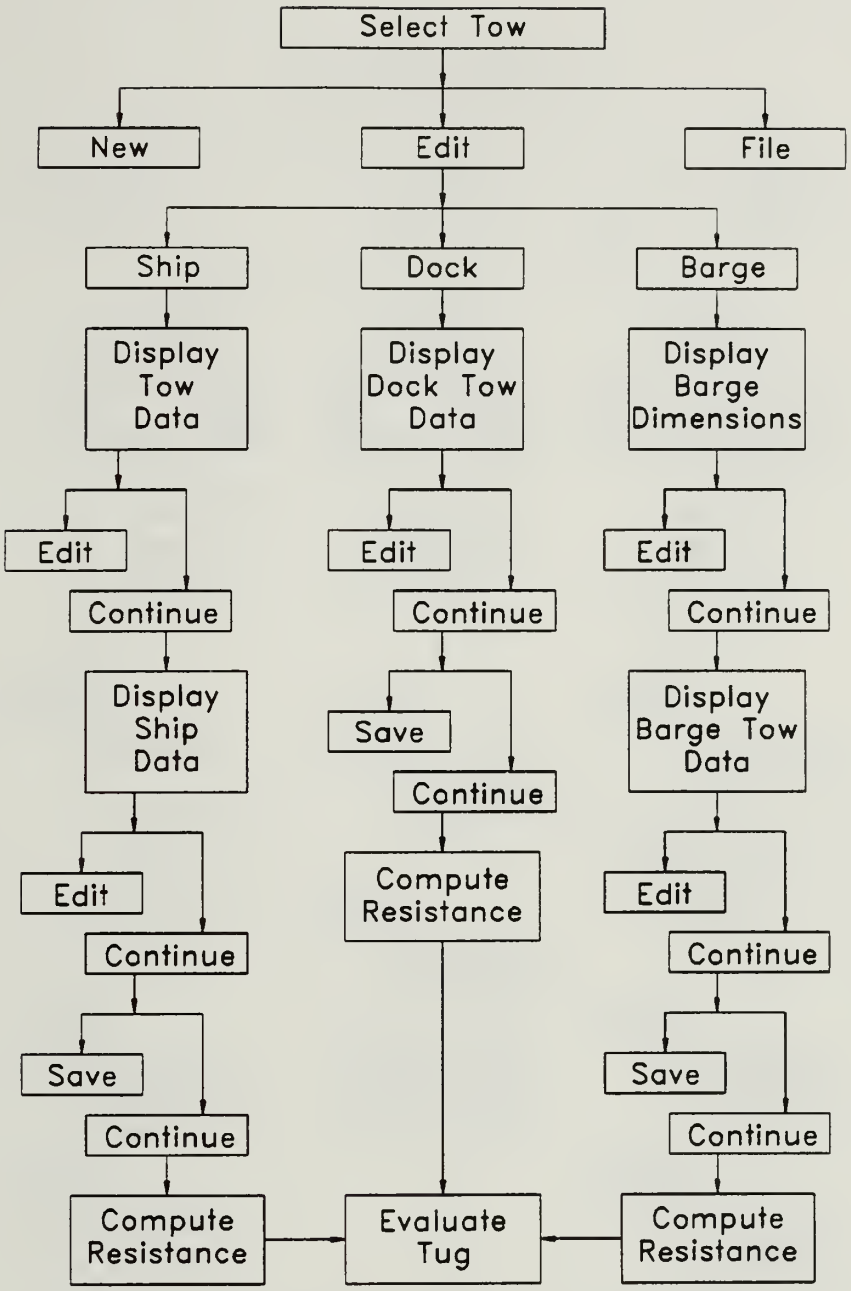

Figure 4.30. Program Flow: Edit Existing Tow Data

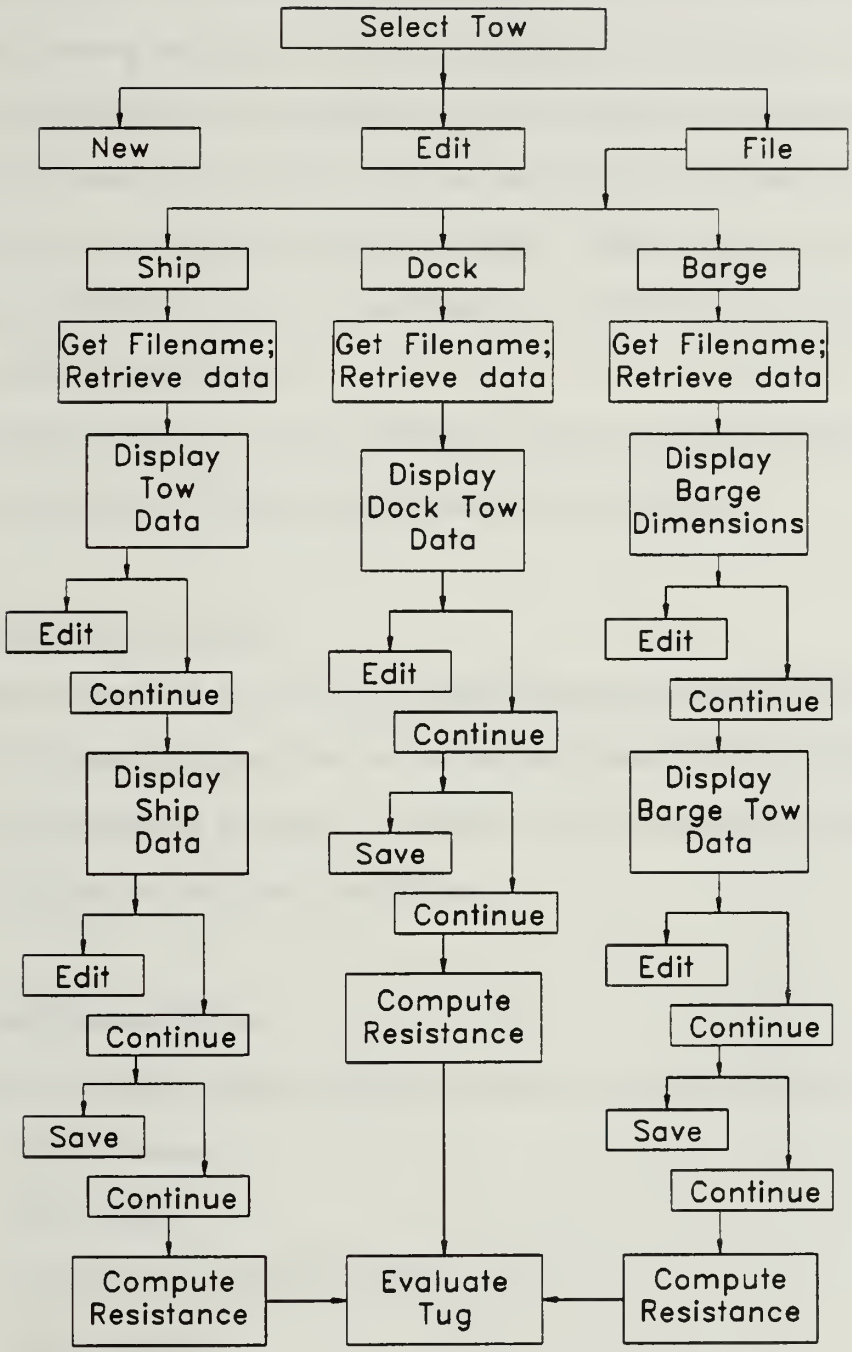

Figure 4.31. Program Flow: Retrieve Tow Data File

If data for a particular tow have been saved to file, that data may be retrieved by choosing **Select Tow** at the **PROGRAM OPTIONS** menu, then selecting **Retrieve data from file**. After choosing the ship type, the user is prompted for a file name. An error message appears if invalid characters are entered, or if the file could not be opened. Additionally, **TOWCALC** checks that the file is a valid tow file (with the extension ".TOW"), and further checks that the data matches the specified ship type. If these conditions are not met, an error message is displayed and the user is prompted to try another file name. Typing "Q" returns the user to the tow options menu.

Provided the tow file is valid, **TOWCALC** reads the data file. From this point, the steps are the same as in **Editing existing data** for each ship type.

4.3.4 Resistance Predictions

When all data for a given tow have been entered and edited to the user's satisfaction, **TOWCALC** computes the resistance of the tow and of the towline at the desired towing speed, using the methods described in Chapter 3. Since the results are presented differently for each ship type, we will look at each in turn.

4.3.4.1 Self-Propelled Ships

For this ship type, a separate summary is displayed, showing (Figure 4.32):

- Wind resistance,
- Wave height,
- Added resistance due to waves,
- Hull resistance,
- Propeller resistance,
- Total tow resistance,
- Hawser resistance,

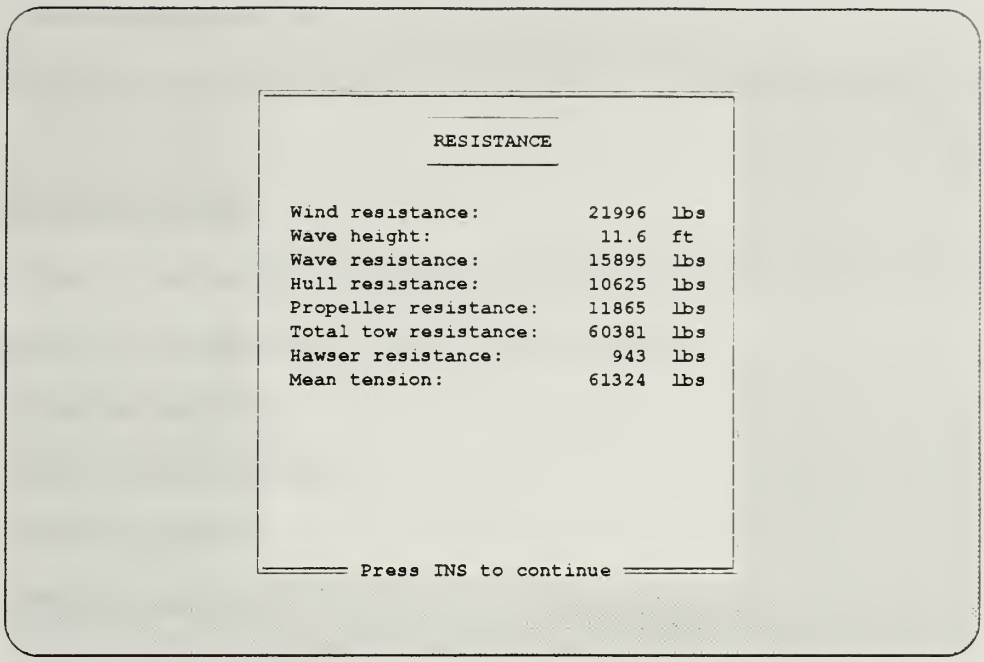

Figure 4.32. Resistance Summary for Self-Propelled Ships

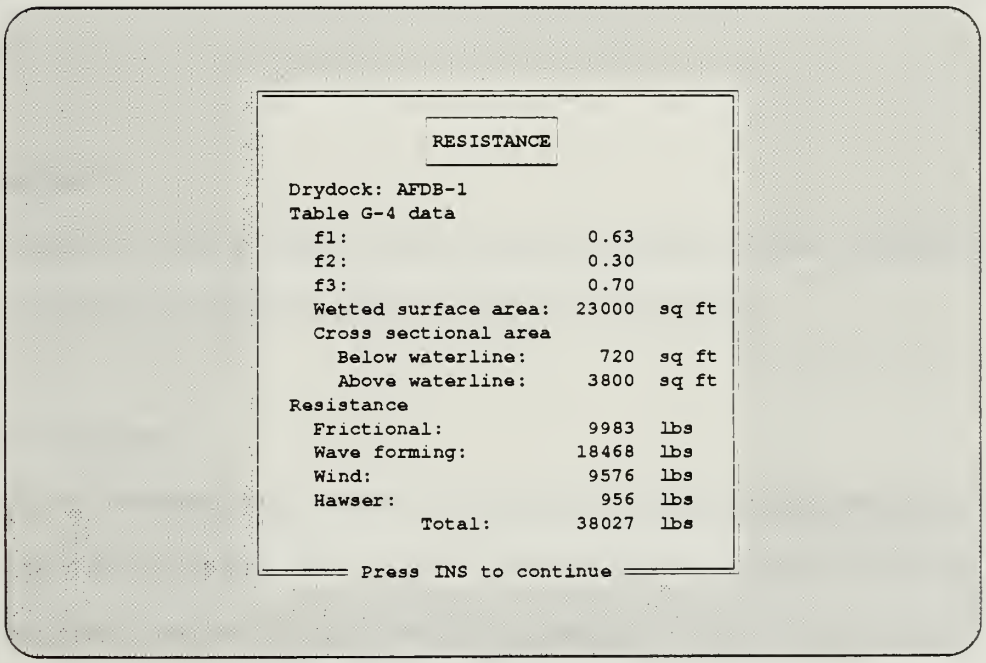

Figure 4.33. Resistance Summary for Floating Drydocks

- Mean towline tension.

When the user is done viewing this screen, pressing Ins continues program execution.

4.3.4.2 Floating Drydocks

For drydocks, the resistance predictions are listed along with a summary of the drydock data (Figure 4.33). The resistance is broken down as follows:

- Frictional resistance,
- Wave forming resistance,
- Wind resistance,
- Hawser resistance,
- Mean towline tension.

Again, when the user is done viewing this screen, pressing Ins continues program execution.

4.3.4.3 Barges

For barges, a summary display similar to that for drydocks is shown (Figure 4.34). The resistance predictions displayed are the same as those for drydocks.

4.3.5 Tug Evaluation

So far, the choice of tug, tow, and tow speed have been made independently. Clearly, this is not possible in practice, since a given tug has a fixed engine plant, with finite power. Part of that power provides thrust to drive the tug through the water at a given speed, and the rest is available for towing. If the mean towline tension at a given tow speed exceeds the available towing thrust, the tug will be unable to make that speed. To address this problem, **TOWCALC** evaluates the ability of the tug to pull the tow at the given speed.

Figure 4.35 gives curves of available thrust versus towing speed for the four classes of Navy towing vessels used in **TOWCALC** [3]. At zero speed, the maximum installed power is available as thrust, the so-called bollard pull condition, while at the ship's maximum free running speed, the available thrust is zero.

After predicting the resistance of the tow, **TOWCALC** evaluates the tug/tow/towspeed combination as follows. First it computes the available thrust (also called available tension) by interpolating the appropriate curve in Figure 4.35 at the desired tow speed. Next, the "best possible" tow speed is found by iteratively computing the tow resistance and towline resistance at each speed (starting at zero), computing the available tension at that speed, and comparing the two. If the available tension is greater than the mean towline tension, the tow speed is increased by one knot. This continues until the available tension is less than the mean tension—the "best" speed is the last speed at which the available tension exceeded the mean tension; this speed may be greater than or less than the desired tow speed. An exact solution is not computed because the accuracy of the resistance prediction methods does not warrant such precision.

RESISTANCE

Name or hull no: YC 1711
Data corresponding to Table G-4

f1:	0.63	
f2:	0.20	
f3:	0.60	
Wetted surface area:	10700	sq ft
Cross sectional area		
Below waterline:	400	sq ft
Above waterline:	600	sq ft
Resistance		
Frictional:	4644	lbs
Wave_forming:	132	lbs
Wind:	951	lbs
Hawser:	2552	lbs
Total:	8279	lbs

Press INS to continue

Figure 4.34. Resistance Summary for Barges

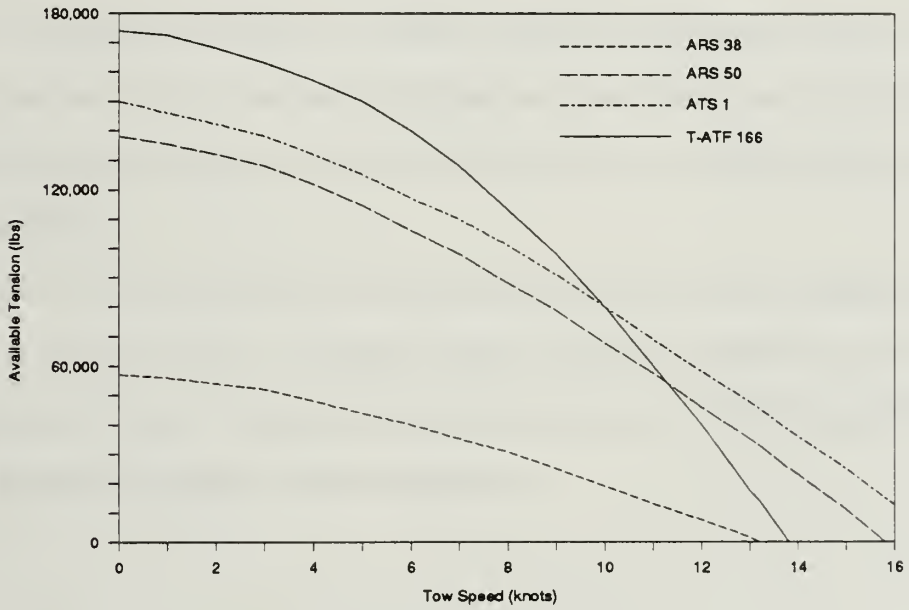

Figure 4.35. Available Tension vs. Tow Speed

After the best speed is found, a summary is displayed (Figure 4.36) with a popup menu giving the following choices:

1) Use best possible speed

2) Use original tow speed

3) Show graph

4) Return to PROGRAM OPTIONS

Choosing option 1) sets the tow speed to the best speed found by **TOWCALC**; choosing 2) retains the original tow speed. If the original tow speed is greater than the best possible speed, an error message is displayed to the effect that the tug has inadequate thrust at this speed, prompting the user to select the best possible speed option (Figure 4.37). After the user selects a valid speed option, **TOWCALC** displays a message confirming the choice (Figure 4.38).

Option 3) displays the curves of available tension vs. towing speed (Figure 4.39) and plots points corresponding to the desired (original) tow speed and the best possible speed. Pressing any key clears the screen and returns the user to the tug evaluation menu, with option 4) highlighted.

If options 3) or 4) are selected before the tow speed is set, an error message is displayed and the user is returned to the tug evaluation menu. Once the tow speed is set, the user may choose either option 3) or 4). Choosing 4) returns the user to the **PROGRAM OPTIONS** menu, with **Estimate Dynamic Tension** highlighted.

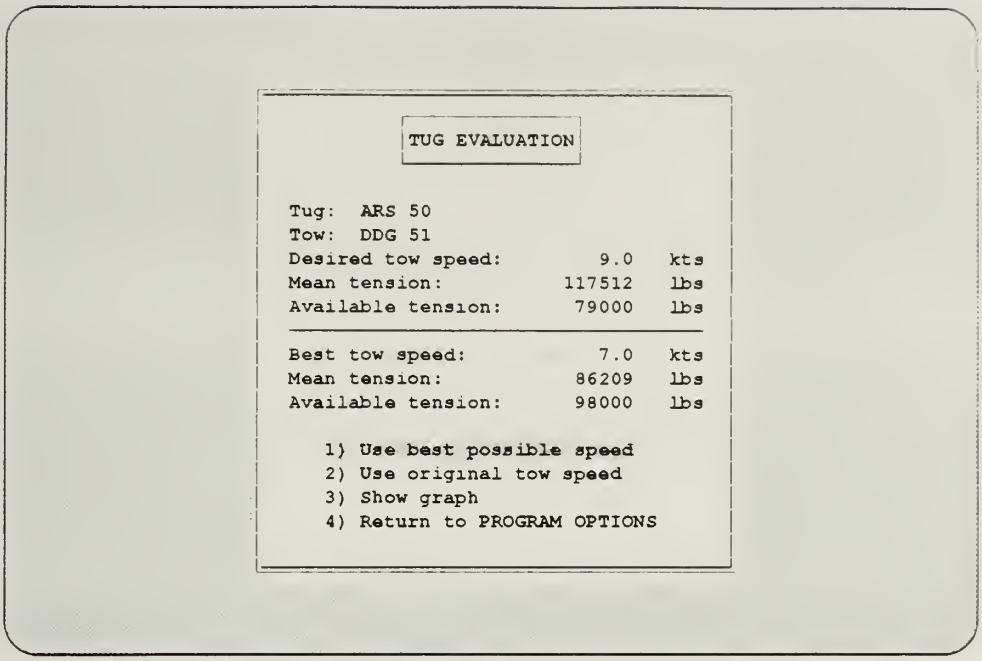
TUG EVALUATION
Tug: ARS 50
Tow: DDG 51
Desired tow speed: 9.0 kts
Mean tension: 117512 lbs
Available tension: 79000 lbs
Best tow speed: 7.0 kts
Mean tension: 86209 lbs
Available tension: 98000 lbs
1) Use best possible speed
2) Use original tow speed
3) Show graph
4) Return to PROGRAM OPTIONS

Figure 4.36. Tug Evaluation Summary and Options Menu

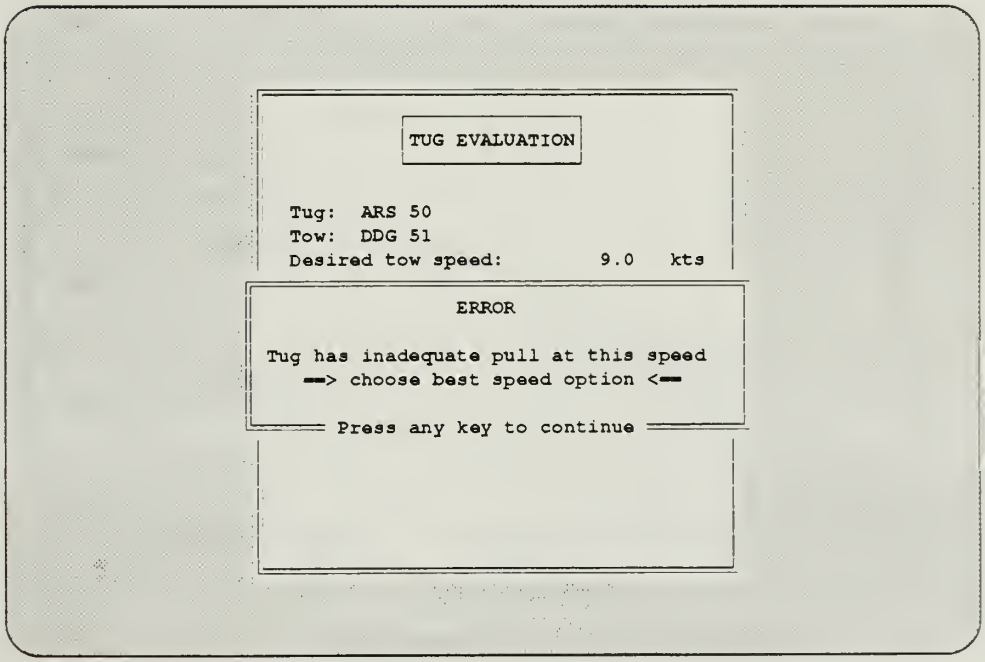
TUG EVALUATION
Tug: ARS 50
Tow: DDG 51
Desired tow speed: 9.0 kts
ERROR
Tug has inadequate pull at this speed
==> choose best speed option <==
Press any key to continue

Figure 4.37. Inadequate Tug Pull Error Message

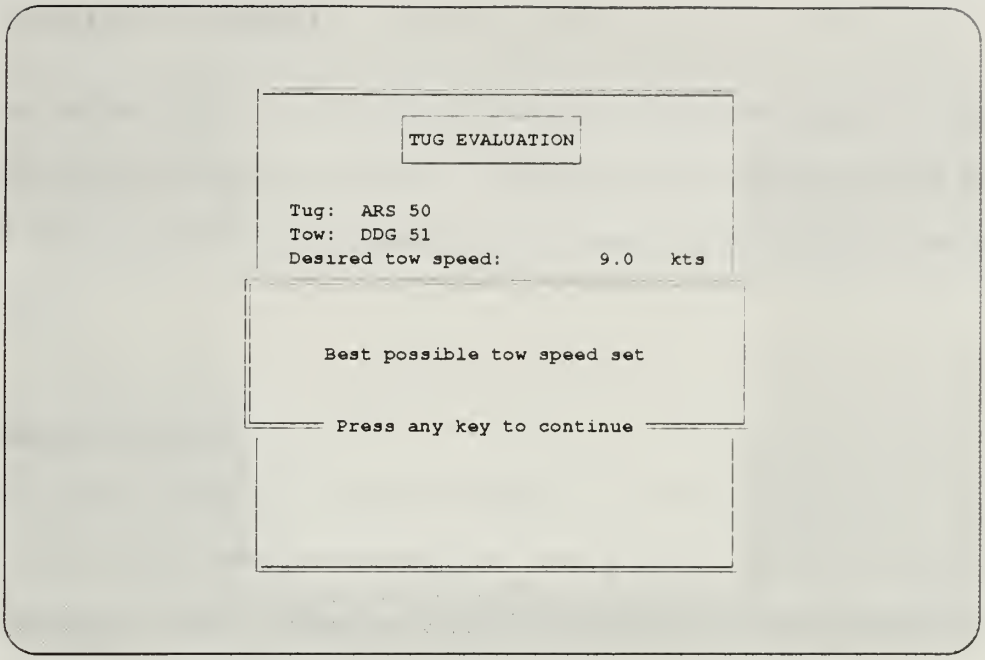

Figure 4.38. Confirmation of Speed Selection

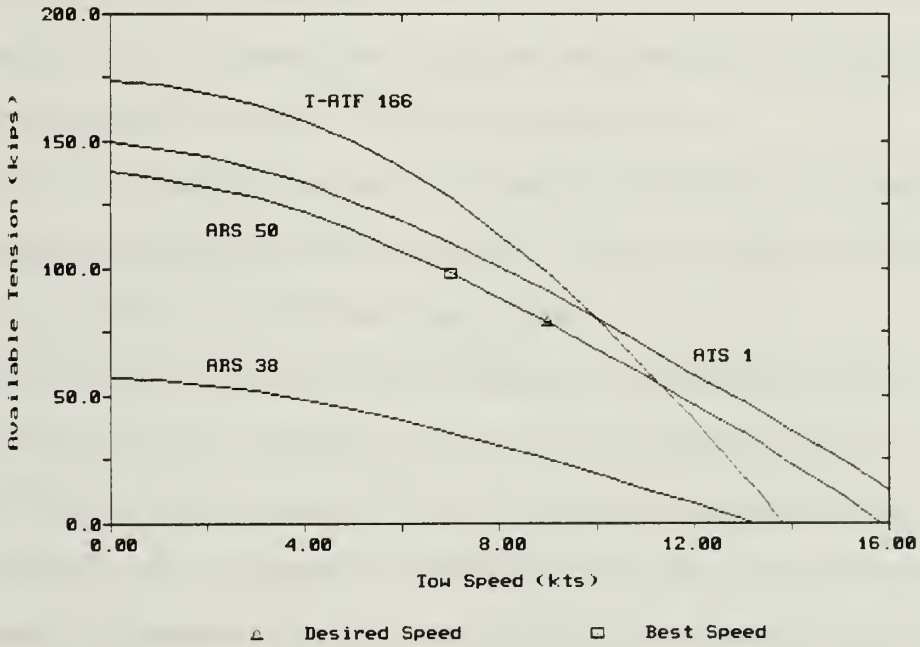

Figure 4.39. Screen Plot of Available Tension vs. Tow Speed

4.4 Extreme Tension

This program option uses the results of Chapter 2 to predict the extreme towline tension for the conditions the user has specified. Additionally, this option shows the user the effects of changing the two most important variables under his or her control: tow speed and towline length.

4.4.1 Using the Data Base

As described in Chapter 2, the extreme tension data base is a collection of "standard curve" choices for each combination of tug, tow, towing speed, wind speed, hawser length, and heading angle. Table 2.1 gives the range of these parameters; the inclusion of CVN 65 data gives 4320 combinations in the data base.

Despite its size, the data base by no means covers all possible cases. Thus, a set of rules is needed to guide in selecting the appropriate combination that most closely matches the conditions specified by the user. We turn to these rules next.

Tug type. Although four tug classes are available in **TOWCALC**, for purposes of extreme tension calculations the ARS 50 and ATS 1 classes are sufficiently close in hydrodynamic characteristics to be considered a single class.

Tow type. Extreme tension is directly influenced by the magnitudes of the relative motions of the tug and tow. Generally, smaller vessels will be affected more at a given sea state than will larger vessels, and thus will have larger relative motions, with correspondingly higher extreme tensions. Consequently, **TOWCALC** selects the tow from the data base with the next smaller displacement, unless the tow's displacement is within 25 percent of the next larger vessel in the data base. This leads to a conservative estimate of extreme tension, that is, one with a higher extreme, but which errs on the side of safety.

Tow speed. Speed contributes to extreme tension in two ways. First, the encounter frequency of waves, for head seas, increases as speed increases, thus raising slightly the added resistance due to waves. Second, and far more significantly, higher tow speeds lead to higher mean towline tensions. This increases extreme tensions by raising the base to which the dynamic tension is added, and also by providing a stiffer towline system (due to reduced catenary depth).

However, since the extreme tension is computed using mean tension as an argument, the choice of tow speed in selecting a standard curve is not a particularly sensitive parameter. Consequently, the following rules are used to select one of the three tow speeds:

$$0 < V_{TOW} < 4.5 \quad : \quad (V_{TOW})_{DataBase} = 3.0 \quad \text{kts}$$

$$4.5 \leq V_{TOW} < 7.5 \quad : \quad (V_{TOW})_{DataBase} = 6.0 \quad \text{kts}$$

$$V_{TOW} \geq 7.5 \quad : \quad (V_{TOW})_{DataBase} = 9.0 \quad \text{kts}$$

Wind speed. TOWCALC uses the following rules to choose the appropriate wind speed in the data base:

$$0 \leq V_{WIND} < 17.5 \quad : \quad (V_{WIND})_{DataBase} = 15.0 \quad \text{kts}$$

$$17.5 \leq V_{WIND} < 22.5 \quad : \quad (V_{WIND})_{DataBase} = 20.0 \quad \text{kts}$$

$$22.5 \leq V_{WIND} < 27.5 \quad : \quad (V_{WIND})_{DataBase} = 25.0 \quad \text{kts}$$

$$27.5 \leq V_{WIND} \leq 48.0 \quad : \quad (V_{WIND})_{DataBase} = 30.0 \quad \text{kts}$$

As noted in section 4.3.1.1, wind speeds above 48 knots generate wave heights outside the range of data in Figure 3.2.

Hawser lengths. Since shorter hawsers result in a stiffer towline system and higher dynamic tensions, **TOWCALC** uses the next shorter length according to the following rules:

$$
\begin{aligned}
L < 1000 &\quad : \quad \text{ERROR} \\
1000 \le L < 1200 &\quad : \quad L_{DataBase} = 1000 \quad \text{ft} \\
1200 \le L < 1500 &\quad : \quad L_{DataBase} = 1200 \quad \text{ft} \\
1500 \le L < 1800 &\quad : \quad L_{DataBase} = 1500 \quad \text{ft} \\
1800 \le L < 2100 &\quad : \quad L_{DataBase} = 1800 \quad \text{ft} \\
L \ge 2100 &\quad : \quad L_{DataBase} = 2100 \quad \text{ft}
\end{aligned}
$$

Open ocean tows are very rarely conducted with less than 1000 feet of hawser, as the danger of breaking the towline in anything but calm seas is too high. Consequently, it is reasonable to restrict hawser length in this manner.

Heading angle. Extreme tensions are not particularly sensitive to heading angle, Γ, and so the following rules are used:

$$
\begin{aligned}
0 \le \Gamma < 40 &\quad : \quad \Gamma_{DataBase} = 0 \quad \text{deg} \\
40 \le \Gamma \le 70 &\quad : \quad \Gamma_{DataBase} = 60 \quad \text{deg} \\
70 < \Gamma < 110 &\quad : \quad \text{ERROR} \\
110 \le \Gamma < 140 &\quad : \quad \Gamma_{DataBase} = 120 \quad \text{deg} \\
140 \le \Gamma \le 180 &\quad : \quad \Gamma_{DataBase} = 180 \quad \text{deg}
\end{aligned}
$$

As discussed in section 4.3.1.1, heading angles between 70 and 110 degrees are not included in the data base because the motions of the tug (particularly rolling) are more likely to limit operations than is extreme towline tension.

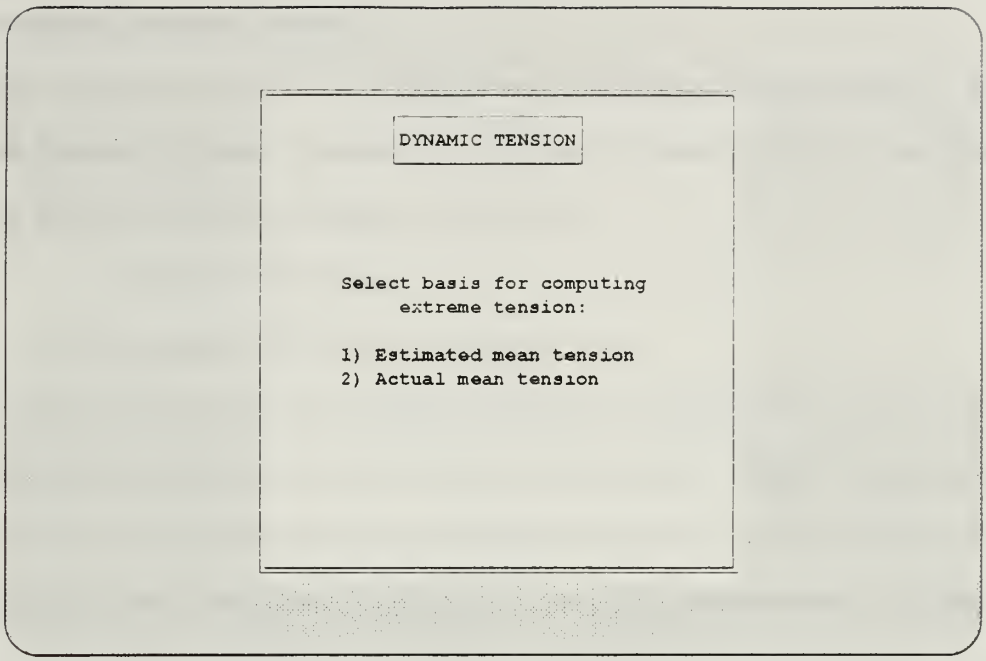

Figure 4.40. Menu for Selecting Dynamic Tension Basis

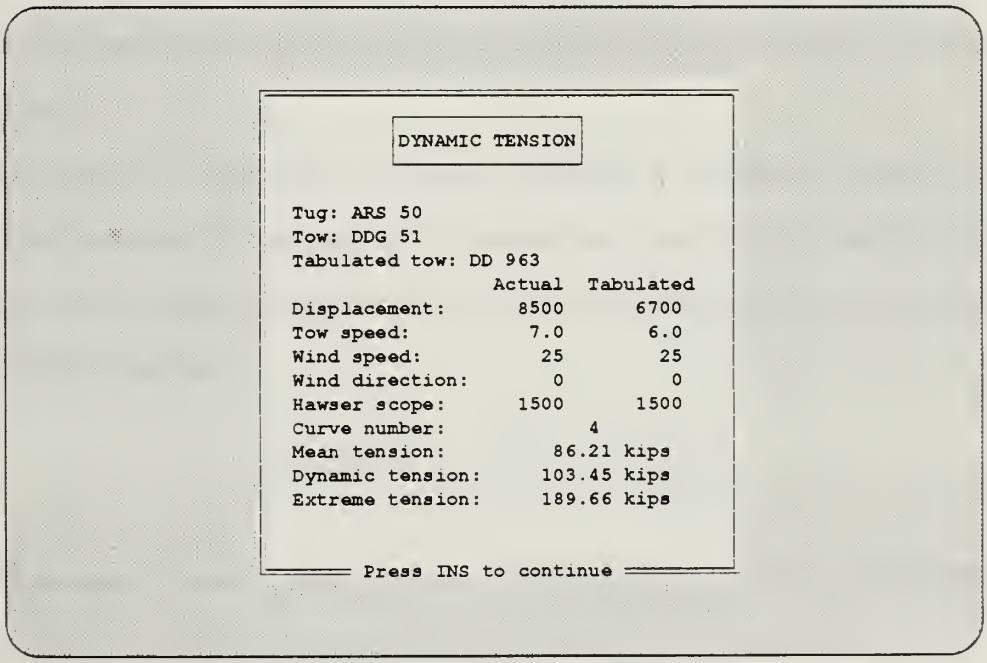

Figure 4.41. Dynamic Tension Summary

4.4.2 Computing Extreme Tension

Figure 4.40 shows schematically the process of estimating dynamic tension. When the user selects **Estimate Dynamic Tension**, a popup menu appears (Figure 4.41) with two choices for the basis of the calculations which follow:

1) Estimated mean tension

2) Actual mean tension

The first option uses the mean tension as estimated in **Select Tow**. The second option allows the user to enter the actual mean tension as determined in a real towing situation. This last gives a far more accurate prediction of extreme tension, but is only useful once the tow is underway, and if there is reasonable confidence in the tension measurement. For planning purposes, the estimated mean tension is the only available data. If the user elects to use a measurement of actual mean tension, **TOWCALC** prompts for its entry. Values above 120 kips are not allowed, as this is the largest mean tension used in computing the extreme tension data base.

Once the basis for calculations is chosen, **TOWCALC** computes the extreme tension. This involves searching the data base for the appropriate standard curve, and then computing the extreme tension using the equation for that curve, with mean tension as the argument. The form of the equation is

$$T_{ext} = 100\left[\overline{T} + \frac{f_a\overline{T}}{(1+f_b\overline{T})} + \frac{f_c\overline{T}^4}{(1+f_d\overline{T}^4)}\right]$$

where $\overline{T}$ is the mean tension in kips divided by 100, and f_a, f_b, f_c, and f_d are coefficients determined by the particular choice of standard curve.

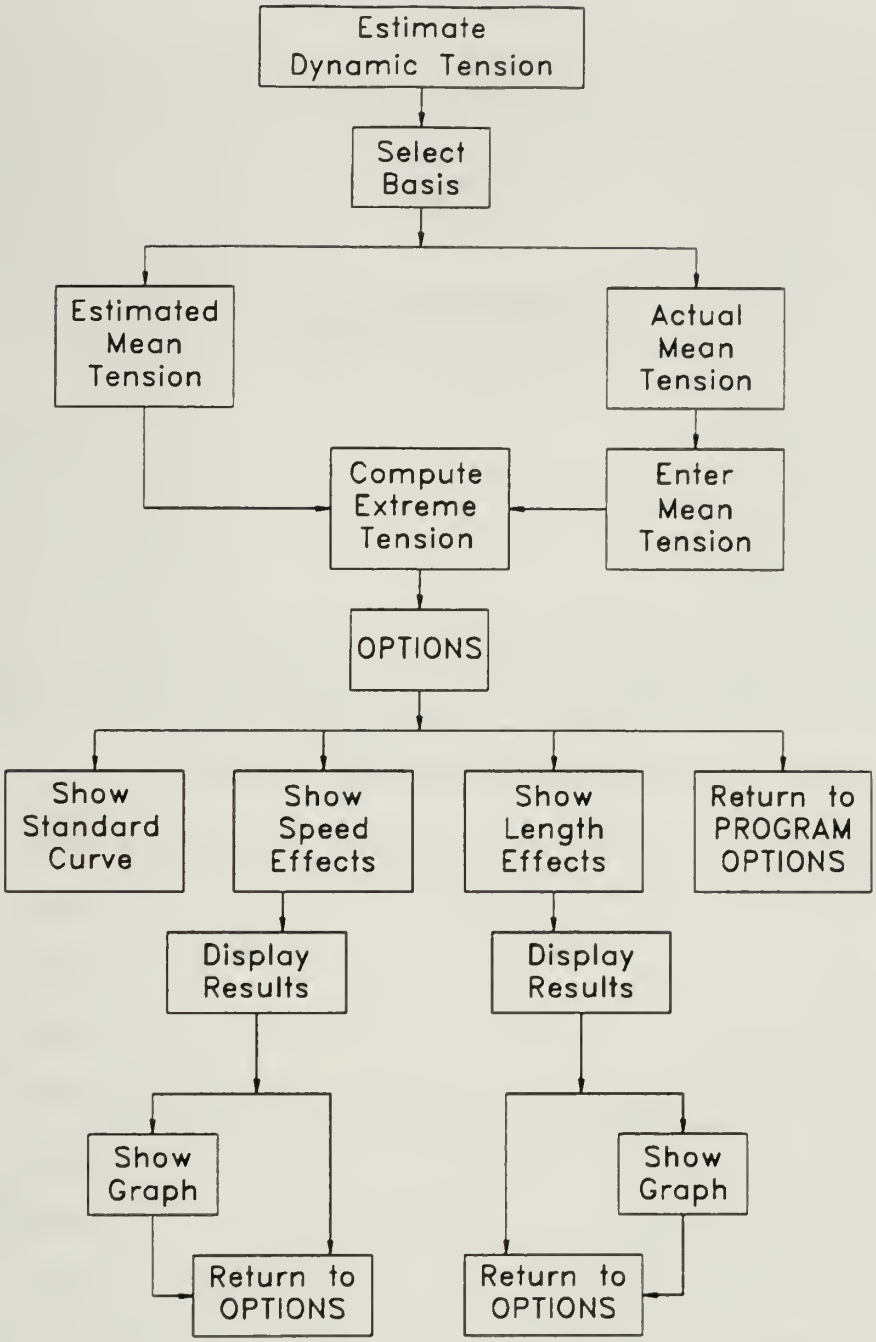

Figure 4.42. Program Flow: Estimate Dynamic Tension

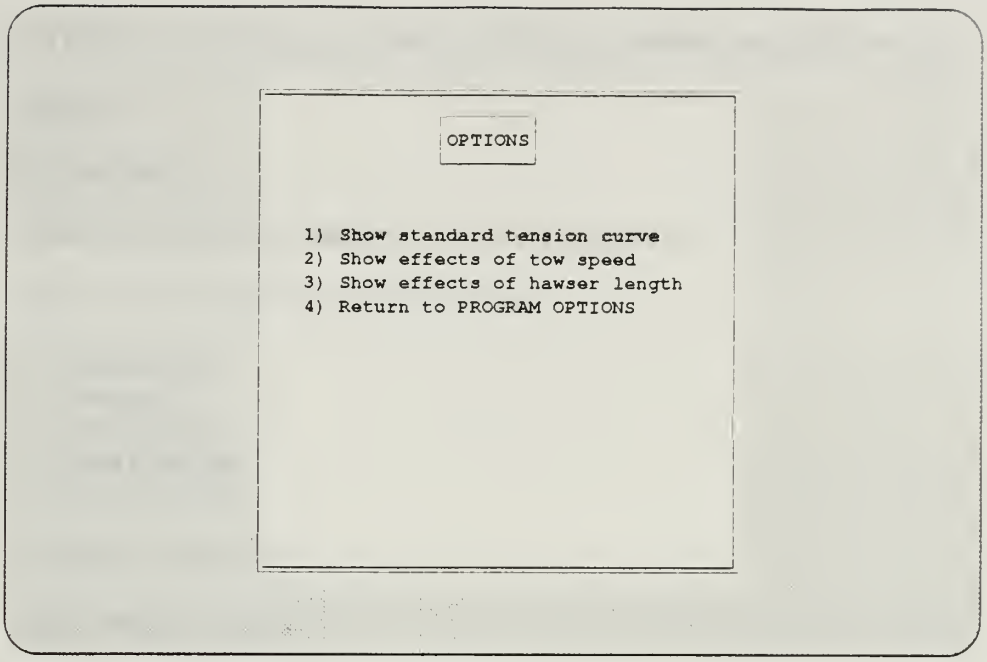
OPTIONS
1) Show standard tension curve
2) Show effects of tow speed
3) Show effects of hawser length
4) Return to PROGRAM OPTIONS

Figure 4.43. Dynamic Tension Options Menu

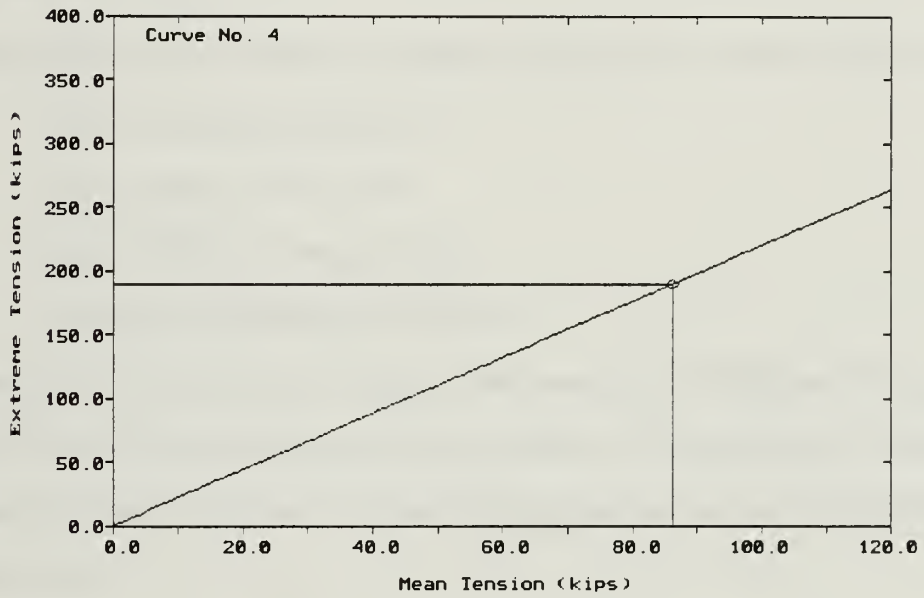

Figure 4.44. Screen Plot of Extreme vs. Mean Towline Tension

Next, **TOWCALC** displays a summary, showing the following (Figure 4.42):

- tug class,
- tow selected,
- closest tow from data base,
- actual and tabulated (from the data base)

  —displacement
  —tow speed
  —wind speed
  —wind direction
  —hawser length

- standard curve number,
- mean towline tension,
- dynamic tension (extreme minus mean),
- extreme tension.

Pressing Ins clears the screen and displays a popup menu of options (Figure 4.43):

1) Show standard tension curve

2) Show effects of tow speed

3) Show effects of hawser length

4) Return to PROGRAM OPTIONS

Selecting the first option plots a graph of the standard curve (Figure 4.44), marked with the particular values of mean tension and extreme tension. Pressing any key clears the screen and returns to the menu of options. The last option returns the user to the **PROGRAM OPTIONS** menu.

We will look at the remaining two options in the next section.

4.4.3 Exploring Alternatives

Of the many parameters which influence extreme towline tension, the tug operator has real control over only two: tow speed, and towline length. Wind and waves are obviously not controllable, although the prudent mariner will avoid severe weather when possible. Heading angle is usually dictated by the course the tug must maintain to complete the journey. Consequently, the only means at the tug operator's disposal to reduce extreme towline tension is to reduce speed, or pay out hawser, or some combination of both.

TOWCALC provides the tug operator, as well as the tow planner, the ability to see the effects of changing both tow speed and towline length. Large changes in either must be made manually, by entering the **Select Tug** or **Select Tow** options, but the effects of incremental changes may be seen by selecting **Show effects of tow speed** or **Show effects of hawser length**. We must note that these options use the estimated mean tension as the basis for their calculations; selecting the actual mean tension as the basis at the beginning of **Estimate Dynamic Tension** does not change this.

4.4.3.1 Effects of Tow Speed

When this option is chosen, **TOWCALC** computes the mean towline tension and extreme tension for tow speeds equal to the original tow speed plus or minus one knot. If the original speed is 1.0 knots, the lower speed is also set to 1.0 knot; if the original speed is 12.0 knots, the upper speed is also set to 12.0 knots.

DYNAMIC TENSION

Tow speed:	6.0	7.0	8.0
Curve number:	4	4	3
Mean tension:	72.7	86.2	100.8
Dynamic tension:	87.2	103.5	90.7
Extreme tension:	159.9	189.7	191.5

1) Show graph
2) Return to OPTIONS

Figure 4.45. Effects of Towing Speed With Options Menu

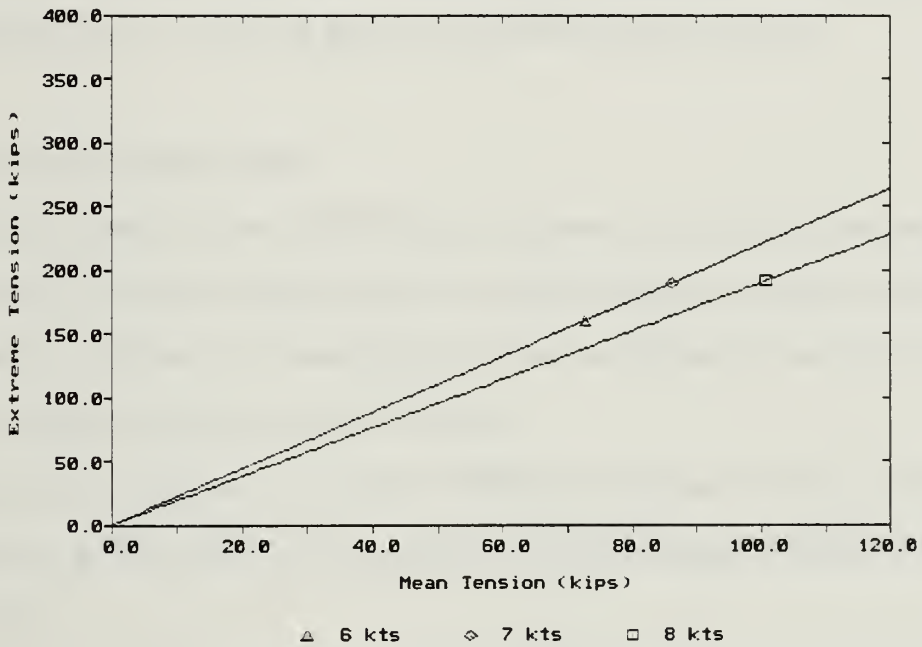

Figure 4.46. Screen Plot of Effects of Towing Speed

Once these calculations are complete, **TOWCALC** displays the results (Figure 4.45), showing:

- tow speed,
- standard curve number,
- mean tension,
- dynamic tension,
- extreme tension.

A popup menu gives the choices:

1) Show graph

2) Return to OPTIONS

Choosing **Show graph** shows the standard tension curves for each speed, marked with the corresponding extreme tension (Figure 4.46); pressing any key clears the screen and returns to the **OPTIONS** menu. Choosing **Return to OPTIONS** skips the graph.

4.4.3.2 Effects of Towline Length

When this option is chosen, **TOWCALC** computes the mean towline tension and extreme tension for towline lengths equal to the original length plus or minus 300 feet. If the original length is 1000 feet, the lower length is also set to 1000 feet; if the original length is 2100 feet, the upper length is also set to 2100 feet.

Once these calculations are complete, **TOWCALC** displays the results, showing the same display as in **Show effects of tow speed**, with towline length substituted for tow speed (Figure 4.47).

The same popup menu appears as in **Show effects of tow speed**, with the same functions. Figure 4.48 shows the extreme vs. mean tension curves for each length, marked with the corresponding extremes.

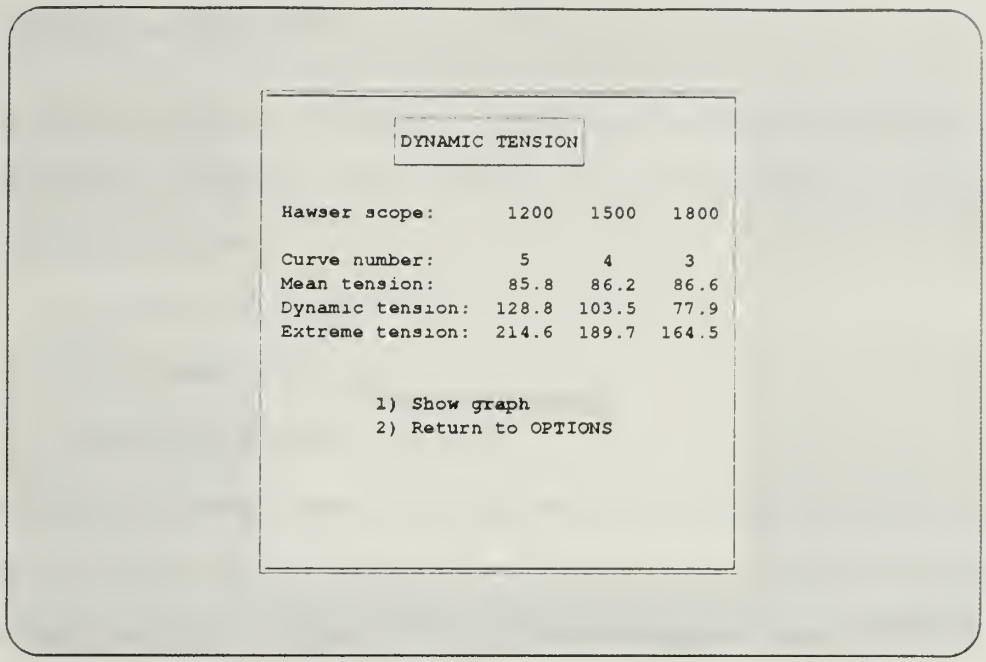
DYNAMIC TENSION
Hawser scope: 1200 1500 1800
Curve number: 5 4 3
Mean tension: 85.8 86.2 86.6
Dynamic tension: 128.8 103.5 77.9
Extreme tension: 214.6 189.7 164.5
1) Show graph
2) Return to OPTIONS

Figure 4.47. Effects of Hawser Length With Options Menu

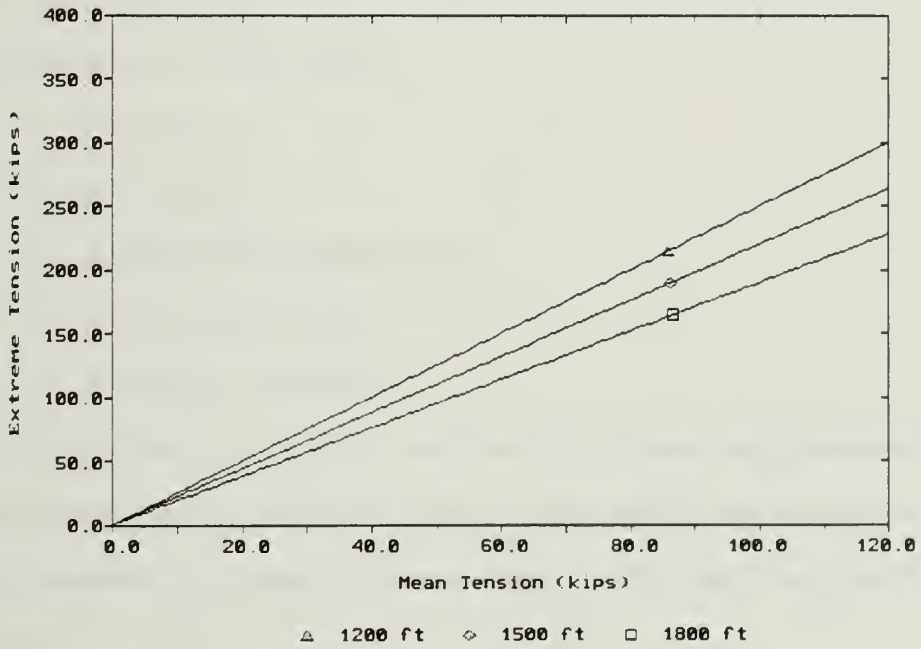

Figure 4.48. Screen Plot of Effects of Hawser Length

4.5 Printing a Report

This option gives the user the ability to obtain hard copy output of the results of a **TOWCALC** session. When this option is chosen, a popup menu is displayed giving the following choices (Figure 4.49):

1) Send report to printer

2) Send report to file only

3) Return to PROGRAM OPTIONS

The first option creates a report by writing data to a file ("REPORT.OUT"), then sending the file to the printer. **TOWCALC** prompts the user to ensure the printer is connected and online before attempting to print. Choosing the second option causes **TOWCALC** to write the report data to a file, but does not send it to the printer. The third option returns the user to the main program menu.

The report itself is in five sections:

I. TUG DATA

II. TOW DATA

III. RESISTANCE PREDICTION

IV. TUG EVALUATION

V. DYNAMIC TENSION

A sample report is given as Appendix B. Each section duplicates the appropriate data summary shown on the screen by **TOWCALC** for the corresponding program option. For example, Section I contains the information shown in Figure 4.6; Section II duplicates Figures 4.17 and 4.19.

Figure 4.49. Print Report Options Menu

Chapter 5

Conclusions

5.1 Summary

The theory of extreme seakeeping loads in towlines makes possible a more accurate evaluation of towing systems than previous practice allowed, leading to safer, more efficient ocean towing. Reducing the uncertainty of the dynamic loads to which towlines are subject makes possible a corresponding reduction in the factors of safety applied to towing system design. When the dynamic tensions are small, this can lead to higher towing speeds, which in turn shortens the length of time spent at sea, reducing exposure to the elements and further diminishing the risk of towline failure.

During an actual tow, the ability to predict dangerous peak tensions gives tug operators the opportunity to take action to reduce those peaks. The most effective means to reduce extreme towline tensions are to reduce speed, increase the length of the hawser, or both.

The computer program which forms the bulk of the work in this thesis, **TOWCALC**, greatly simplifies the task of tow planning by automating the process of predicting mean and extreme towline tensions. By doing this in a user-friendly manner, **TOWCALC** gives both planners and operators the freedom to explore different alternatives rapidly and efficiently. In so doing, it will also help tug operators gain a better understanding of the actions they can take at sea to reduce peak dynamic tensions.

I must emphasize, however, that **TOWCALC** is <u>not</u> an expert system. Its primary purpose is to serve as a computational tool to aid in decision making, but not to make those decisions itself. The final evaluation of a given tow design is left, properly, to the user.

5.2 Further Study

Although a great deal has been accomplished in the study of towing dynamics, there remains much to be done.

An important first step is to validate the theory of extreme tensions with full-scale experiments. A test plan for such experiments has been developed (Christensen [10]), and a full-scale trial has been scheduled by the Navy.

More research is needed to develop a theory to better predict the added resistance in waves at the low speeds used in towing. This is the largest source of uncertainty in the prediction of mean tension, and any improvements in accuracy would greatly improve the utility of **TOWCALC** as a planning tool.

Expanding the current data base of extreme tension results to include more ship types, longer towing hawsers, and different configurations of chain and hawser would improve the overall accuracy of the predictions. This should be combined with a thorough sensitivity analysis, based on a parametric hull form, to gain better insight into the effects of hull geometry on extreme tensions.

There are many improvements which could be made to the program **TOWCALC**, and Fleet use will inevitably suggest a number of such improvements. One important area which has been left out is the theory of extreme tensions for fiber rope towlines [2]; although only the newer classes of Navy towing vessels are able to handle large circumference synthetic lines, adding this capability to **TOWCALC** would improve the flexibility of the program. Despite this omission, I believe that **TOWCALC** in its present form is a highly useful tool.

Bibliography

[1] Frimm, F.C., *Non-Linear Extreme Tension Statistics of Towing Hawsers*, PhD Thesis, MIT Dept. of Ocean Engineering, Cambridge, Mass., 1987

[2] Milgram, J.H., Triantafyllou, M.S., Frimm, F.C., Anagnostou, G., "Seakeeping and Extreme Tensions in Offshore Towing," paper submitted for presentation at Society of Naval Architects and Marine Engineers Annual Meeting, November 1987

[3] Naval Sea Systems Command, *U.S. Navy Towing Manual*, September 1988

[4] Triantafyllou, M.S., *Nonlinear Dynamics of Marine Cables and Hawsers*, Technical Report, MIT Dept. of Ocean Engineering, Cambridge, Mass., 1987

[5] Newman, J.N., *Marine Hydrodynamics*, MIT Press, Cambridge, Mass., 1977

[6] Berteaux, H.O., *Buoy Engineering*, John Wiley & Sons, New York, N.Y., 1976

[7] Doerry, N.H., *C Input-Output Routines*, Private communication with the author, MIT Dept. of Ocean Engieering, Cambridge, Mass., 1989

[8] Milgram, J.H., *Plotting Routines for the Athena Lab Computer*, Private communication with the author, MIT Dept. of Ocean Engieering, Cambridge, Mass., 1987

[9] Schildt, H., *C: Power User's Guide*, Osborne McGraw-Hill, Berkeley, Calif., 1988

[10] Christensen, E.N., *Plans and Specifications for a Full-Scale Towing Model Validation Experiment*, Naval Engineer Thesis, MIT Dept. of Ocean Engineering, Cambridge, Mass., 1989

Appendix A

Extreme Tension Data for CVN 65

Tables A.1 through A.3 give the results of the extreme tension calculations for the nuclear-powered aircraft carrier CVN 65 towed by ARS 38, ARS 50/ATS 1, and T-ATF 166 respectively.

Tug ARS-38 Towing CVN-65 — CURVE NUMBERS FOR VARIOUS TOWLINE LENGTHS (LENGTH-CURVE #)

HEADING ANGLE	15 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000-26 1200-26 1500- 0 1800- 0 2100- 0	1000-26 1200- 0 1500- 0 1800- 0 2100- 0	1000- 0 1200- 0 1500- 0 1800- 0 2100- 0
60	1000-26 1200-26 1500-26 1800- 0 2100- 0	1000-26 1200-26 1500- 0 1800- 0 2100- 0	1000-26 1200- 0 1500- 0 1800- 0 2100- 0
120	1000-26 1200-26 1500-26 1800-26 2100-26	1000-76 1200-26 1500-26 1800-26 2100-26	1000-51 1200-76 1500-26 1800-26 2100-26
180	1000-76 1200-26 1500-26 1800-26 2100-26	1000-51 1200-76 1500- 1 1800-26 2100-26	1000-52 1200-77 1500-51 1800-76 2100- 1
	20 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000- 2 1200-51 1500-27 1800- 1 2100-26	1000-51 1200- 1 1500- 1 1800-26 2100-26	1000- 1 1200- 1 1500-26 1800-26 2100-26
60	1000-28 1200- 1 1500-27 1800-26 2100-26	1000-51 1200- 1 1500-27 1800-26 2100-26	1000-27 1200- 1 1500-26 1800-26 2100-26
120	1000- 2 1200-28 1500-27 1800-27 2100-27	1000-52 1200- 2 1500-28 1800-27 2100-27	1000- 3 1200-52 1500- 2 1800-28 2100-27
180	1000- 4 1200- 3 1500-29 1800-28 2100-28	1000- 6 1200- 4 1500-78 1800-30 2100-29	1000-13 1200- 8 1500- 5 1800- 4 2100- 3
	25 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000- 5 1200- 4 1500-30 1800-29 2100-28	1000- 4 1200- 3 1500-29 1800- 2 2100-28	1000- 3 1200-52 1500-29 1800-28 2100-27
60	1000-31 1200-30 1500-29 1800-28 2100-28	1000-30 1200-29 1500-29 1800-28 2100-27	1000-30 1200-29 1500-28 1800-28 2100-27
120	1000- 5 1200-31 1500-30 1800-29 2100-28	1000- 6 1200- 4 1500-31 1800-30 2100-29	1000- 7 1200- 5 1500-32 1800-30 2100-29
180	1000-40 1200- 7 1500-34 1800-32 2100-31	1000-17 1200-12 1500-37 1800-35 2100-33	1000-21 1200-18 1500-14 1800- 9 2100-36
	30 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000-14 1200- 8 1500-35 1800-33 2100-31	1000-39 1200- 6 1500-33 1800-32 2100-30	1000-37 1200- 5 1500-32 1800-31 2100-29
60	1000- 8 1200-34 1500-32 1800-30 2100-30	1000-36 1200-33 1500-31 1800-30 2100-29	1000-35 1200-32 1500-31 1800-30 2100-29
120	1000-65 1200- 7 1500-33 1800-32 2100-30	1000-15 1200- 8 1500-34 1800-33 2100-31	1000-16 1200-12 1500-36 1800-34 2100-32
180	1000-19 1200-16 1500-41 1800-38 2100-35	1000-22 1200-19 1500-16 1800-12 2100-38	1000-23 1200-74 1500-19 1800-18 2100-43

Figure A.1. ARS 38 Towing CVN 65

TUG ATS-1 TOWING CVN-65 — CURVE NUMBERS FOR VARIOUS TOWLINE LENGTHS (LENGTH-CURVE #)

HEADING ANGLE	15 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000-26 1200-26 1500- 0 1800- 0 2100- 0	1000-26 1200- 0 1500- 0 1800- 0 2100- 0	1000-26 1200- 0 1500- 0 1800- 0 2100- 0
60	1000-26 1200-26 1500-26 1800- 0 2100- 0	1000-26 1200-26 1500-26 1800- 0 2100- 0	1000-26 1200-26 1500-26 1800- 0 2100- 0
120	1000-76 1200-26 1500-26 1800-26 2100-26	1000-51 1200-76 1500-26 1800-26 2100-26	1000-77 1200-51 1500-76 1800-26 2100-26
180	1000-76 1200-26 1500-26 1800-26 2100- 0	1000-51 1200-76 1500-76 1800-26 2100-26	1000-78 1200-77 1500-51 1800-76 2100- 1
	20 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000-52 1200- 2 1500-51 1800-76 2100- 1	1000- 2 1200-51 1500-51 1800- 1 2100-26	1000-28 1200-51 1500- 1 1800-26 2100-26
60	1000-52 1200- 2 1500-51 1800- 1 2100- 1	1000- 2 1200-28 1500-27 1800- 1 2100-26	1000- 2 1200-51 1500-27 1800- 1 2100-26
120	1000- 3 1200-52 1500- 2 1800-51 2100-76	1000- 4 1200- 3 1500-77 1800-28 2100-51	1000- 5 1200- 4 1500- 3 1800-77 2100- 2
180	1000- 5 1200-53 1500-52 1800-77 2100-51	1000-83 1200-55 1500-79 1800-78 2100-77	1000-65 1200-60 1500-81 1800-54 2100-53
	25 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000- 8 1200- 6 1500-79 1800-78 2100-77	1000- 6 1200-79 1500-78 1800-52 2100- 2	1000- 5 1200-53 1500-52 1800-77 2100-51
60	1000-35 1200- 4 1500- 3 1800-29 2100-29	1000-34 1200-32 1500-30 1800-29 2100-28	1000-33 1200-31 1500-30 1800-29 2100-28
120	1000- 8 1200- 6 1500- 4 1800- 3 2100-52	1000-11 1200- 7 1500-54 1800-53 2100- 3	1000-14 1200- 8 1500-55 1800-79 2100-78
180	1000-16 1200-13 1500- 7 1800-80 2100-79	1000-69 1200-16 1500-12 1800-82 2100-80	1000-72 1200-69 1500-16 1800-61 2100-83
	30 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000-18 1200-15 1500- 9 1800-81 2100-54	1000-16 1200-12 1500- 7 1800- 5 2100- 4	1000-14 1200- 8 1500- 6 1800-79 2100- 3
60	1000-67 1200- 9 1500-35 1800-33 2100-31	1000-15 1200-38 1500-34 1800-32 2100-31	1000-13 1200-37 1500-34 1800-32 2100-31
120	1000-67 1200-14 1500- 7 1800- 5 2100- 4	1000-68 1200-15 1500- 8 1800- 6 2100- 5	1000-69 1200-66 1500-12 1800- 7 2100-80
180	1000-72 1200-69 1500-15 1800-12 2100-82	1000-74 1200-71 1500-18 1800-15 2100-12	1000-24 1200-73 1500-70 1800-91 2100-15

Figure A.2. ARS 50/ATS 1 Towing CVN 65

Tug TATF towing CVN-65 — CURVE NUMBERS FOR VARIOUS TOWLINE LENGTHS (LENGTH-CURVE #)

HEADING ANGLE	15 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000-26 1200-26 1500- 0 1800- 0 2100- 0	1000-26 1200-26 1500- 0 1800- 0 2100- 0	1000-26 1200- 0 1500- 0 1800- 0 2100- 0
60	1000-26 1200-26 1500-26 1800- 0 2100- 0	1000-26 1200-26 1500-26 1800- 0 2100- 0	1000-26 1200-26 1500- 0 1800- 0 2100- 0
120	1000-26 1200-26 1500-26 1800-26 2100-26	1000-76 1200- 1 1500-26 1800-26 2100-26	1000-51 1200-76 1500-76 1800-26 2100-26
180	1000-76 1200-26 1500-26 1800-26 2100-26	1000-77 1200-51 1500-76 1800-26 2100-26	1000-78 1200-77 1500-51 1800-76 2100- 1
	20 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000-52 1200- 2 1500-51 1800- 1 2100- 1	1000- 2 1200-51 1500-76 1800- 1 2100-26	1000-51 1200- 1 1500- 1 1800-26 2100-26
60	1000- 2 1200-28 1500-27 1800- 1 2100-26	1000-28 1200-27 1500-27 1800- 1 2100-26	1000-28 1200-27 1500-27 1800-26 2100-26
120	1000- 3 1200-29 1500-28 1800-27 2100- 1	1000-53 1200-30 1500- 2 1800-28 2100-51	1000-79 1200- 3 1500-52 1800- 2 2100-51
180	1000- 5 1200-53 1500-52 1800- 2 2100-28	1000- 8 1200- 6 1500- 4 1800- 3 2100-77	1000-15 1200-12 1500-81 1800-54 2100-53
	25 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000- 7 1200- 5 1500-53 1800-52 2100- 2	1000- 5 1200- 4 1500- 3 1800-77 2100-28	1000- 4 1200- 3 1500-52 1800- 2 2100-51
60	1000-33 1200-31 1500-30 1800-29 2100-28	1000-32 1200-31 1500-29 1800-29 2100-28	1000-32 1200-30 1500-29 1800-28 2100-28
120	1000- 6 1200- 5 1500- 3 1800-30 2100- 2	1000- 8 1200- 6 1500- 4 1800- 3 2100-52	1000-12 1200- 7 1500- 5 1800-53 2100- 3
180	1000-15 1200-10 1500- 6 1800-54 2100- 4	1000-18 1200-15 1500-10 1800- 7 2100- 5	1000-20 1200-68 1500-15 1800-12 2100- 8
	30 KNOT WIND - 3 KNOTS APPROX TOW SPEED	6 KNOTS APPROXIMATE TOW SPEED	9 KNOTS APPROXIMATE TOW SPEED
0	1000-16 1200-13 1500- 7 1800- 5 2100- 4	1000-14 1200- 9 1500- 6 1800- 4 2100-31	1000-10 1200- 7 1500- 5 1800- 4 2100-30
60	1000-64 1200- 7 1500-33 1800-32 2100-31	1000-14 1200-36 1500-33 1800-31 2100-30	1000- 9 1200-35 1500-32 1800-31 2100-30
120	1000-66 1200-13 1500- 6 1800-33 2100-32	1000-67 1200-64 1500- 7 1800- 5 2100- 4	1000-68 1200-65 1500- 8 1800- 6 2100- 5
180	1000-20 1200-18 1500-14 1800- 9 2100- 7	1000-22 1200-70 1500-17 1800-14 2100-10	1000-24 1200-73 1500-69 1800-18 2100-15

Figure A.3. T-ATF 166 Towing CVN 65

Appendix B

Sample Tow Report

This appendix contains a sample tow report produced by the **Print Report** option in **TOWCALC**. The data is for ARS 50 towing DDG 51.

```
*--------------*
|  TOW REPORT  |
*--------------*

-------------------------- S E C T I O N  I --------------------------
------------------------------ TUG DATA ------------------------------

Class: ARS 50

Hawser data
  diameter:      2.25 in
  scope:         1500 ft

Chain pendant data
  size:          2.25 in
  scope:           90 ft

-------------------------- S E C T I O N  II -------------------------
------------------------------ TOW DATA ------------------------------

Tow type:        SHIP

Input data
  Hull number: DDG 51
  Full load displacment:              8500 tons
  Tow speed:                           7.0 kts
  Maximum expected wind speed:        25.0 kts
  Relative wind direction:             0.0 deg
  Propeller status:                Trailing

Table G-2 data
  Ship class: DDG 51-53
  Tabulated displacement:             8300 tons
  Frontal area:                       6900 sq ft
  Wind coefficient:                   0.70
  Propeller area:                      254 sq ft
  Hull resistance curve:                 4
  Wave resistance curve:                 1
```

```
------------------------- S E C T I O N  III -------------------------
------------------------ RESISTANCE PREDICTION ------------------------

Tow type:       SHIP

Wind resistance:                21996 lbs
Wave height:         11.6 ft
Wave resistance:                15895 lbs
Hull resistance:                10625 lbs
Propeller resistance:           11865 lbs
                               --------
Total tow resistance:           60381 lbs

Hawser resistance:                   1848 lbs

Mean towline tension:               62229 lbs  or  62.23 kips


-------------------------- S E C T I O N  IV --------------------------
--------------------------- TUG EVALUATION ----------------------------


Tug: ARS 50
Tow: DDG 51

Desired tow speed:                    5.0 kts
Mean tension at this speed:         61324 lbs
Available tension at this speed:   114700 lbs

            *** Tug has sufficient pull at this speed ***

Best possible tow speed:              7.0 kts
Mean tension at this speed:         86209 lbs
Available tension at this speed:    98000 lbs

       Option selected:
            Tow speed =  7.0 kts (BEST POSSIBLE)
```

---------- S E C T I O N V ----------
---------- DYNAMIC TENSION ----------

Tug: ARS 50
Tow: DDG 51

Tabulated data

Tow: DD 963	
Displacement:	6700 tons
Tow speed:	6 kts
Wind speed:	25 kts
Wind direction:	0 deg
Hawser scope:	1500 ft
Curve number from Appendix O:	4

Results

Mean tension:	86.21 kips (ESTIMATED)
Dynamic tension:	103.45 kips
Extreme tension:	189.66 kips

Note: The following results are based on predicted, not actual tow resistance

Effects of tow speed:

Tow speed (kts):	6.0	7.0	8.0
Curve number:	4	4	3
Mean tension (kips):	72.7	86.2	100.8
Dynamic tension (kips):	87.2	103.5	90.7
Extreme tension (kips):	159.9	189.7	191.5

Effects of hawser scope:

Hawser scope (ft):	1200	1500	1800
Curve number:	5	4	3
Mean tension (kips):	85.8	86.2	86.6
Dynamic tension (kips):	128.8	103.5	77.9
Extreme tension (kips):	214.6	189.7	164.5

Appendix C

Installing TOWCALC

TOWCALC is provided on a single 360 kilobyte floppy disk. Included is an installation program, "setup.exe", which performs all necessary tasks to get **TOWCALC** running on a computer. To install **TOWCALC**, first ensure approximately 500 kilobytes of memory are available on the hard disk. Insert the floppy disk in drive A, and type "A:" at the DOS prompt. Type "setup" and follow the instructions given by the program. When finished, "setup" exits with the current directory as "C:\TOWCALC". To enter **TOWCALC**, type "towcalc".

The source code for "setup" is included as part of this appendix.

```
/*****************************************************************
  File: setup.c
  Author: Todd J. Peltzer
  Last update: 3 May 1989

  This file prepares the user's computer for installation of
  TOWCALC and loads the necessary files.
*****************************************************************/
#include "stdio.h"
#include "dos.h"

static char *label[] =
{
  "                          TOWING CALCULATIONS",
  "                             Setup Program",
  "                This program prepares TOWCALC for use",
  "                on your computer. Please  follow  the",
  "                directions that are given . . . .",
  "                Press any key to continue . . . . ",
  "                . . . . please wait . . . .",
  "                . . . . copying files . . . .",
  "                . . . . cleaning up . . . .",
  "Which graphics display does your computer have?",
  "       1) CGA : Color Graphics Array",
  "       2) EGA : Enhanced Graphics Array",
  "       3) Hercules : Hercules graphics adapter",
  "Enter number of your graphics display: ",
  "                . . . . cleaning up . . . .",
  "To run TOWCALC, change directories to",
  "                c:\\TOWCALC",
  "and type TOWCALC."
};

void wait();

main()
{
  FILE *out;
  int i;
  int result;
  char sys_call[81];
  char device[13];
  int mode;

  /* print intial message on screen */
  result = system("cls");
  printf("\n\n%s\n\n",label[0]);
  printf("%s\n\n",label[1]);
  printf("%s\n",label[2]);
  printf("%s\n",label[3]);
  printf("%s\n\n\n",label[4]);
  printf("%s\n",label[5]);

  wait();

  /* create directory on hard drive */
  sprintf(sys_call,"mkdir c:\\towcalc");
  printf("\n%s\n",label[6]);
  result = system(sys_call);

  /* copy archive to TOWCALC's directory */
  printf("\n%s\n\n",label[7]);
  sprintf(sys_call,"copy a:towfiles.exe c:\\towcalc");
  result = system(sys_call);
```

```
/* extract files from archive */
result = system("c:");
sprintf(sys_call,"cd c:\\towcalc");
result = system(sys_call);
sprintf(sys_call,"towfiles");
result = system(sys_call);

/* delete archive from hard drive */
printf("\n%s\n",label[8]);
sprintf(sys_call,"del towfiles.exe");
result = system(sys_call);

/* get graphics device */
result = system("cls");
while (1)
{
  for (i=0; i<5; i++)
    printf("%s\n",label[9+i]);
  scanf("%d",&i);
  if (i==1)
  {
    sprintf(device,"HALOIBMG.DEV");
    mode=1;
    break;
  }
  else if (i==2)
  {
    sprintf(device,"HALOIBME.DEV");
    mode=4;
    break;
  }
  else if (i==3)
  {
    sprintf(device,"HALOHERC.DEV");
    mode=0;
    break;
  }
  else continue;
}

/* clean up extraneous HALO device drivers */
if (i==1)
{
  printf("\n%s\n",label[14]);
  sprintf(sys_call,"del HALOIBME.DEV");
  result = system(sys_call);
  sprintf(sys_call,"del HALOHERC.DEV");
  result = system(sys_call);
}
else if (i==2)
{
  printf("\n%s\n",label[14]);
  sprintf(sys_call,"del HALOIBMG.DEV");
  result = system(sys_call);
  sprintf(sys_call,"del HALOHERC.DEV");
  result = system(sys_call);
}
else if (i==3)
{
  printf("\n%s\n",label[14]);
  sprintf(sys_call,"del HALOIBMG.DEV");
  result = system(sys_call);
  sprintf(sys_call,"del HALOIBME.DEV");
  result = system(sys_call);
}
```

```
  /* create file DEVICE.DAT */
  out=fopen("DEVICE.DAT","w");
  fprintf(out,"%s\n%d\n",device,mode);
  fclose(out);

  /* print final message */
  result = system("cls");
  printf("\n\n%s\n",label[15]);
  printf("\n%s\n",label[16]);
  printf("\n%s\n",label[17]);
}

void wait()
{
  union inkey
  {
    char ch[2];
    int i;
  } c;

  /* wait for keystroke */
  while (1)
  {
    c.i = bioskey(0);        /* read the key */

    if (c.ch[0])             /* key is a normal key */
    {
      break;
    }
    else                     /* key is a special key */
    {
      break;
    }
  }
}
```

Appendix D

TOWCALC Source Code

This appendix contains the source code for **TOWCALC**. There are eighteen files listed: one "make" file, three include files, and fourteen source files.

The make file is used to compile the program using the Microsoft "make.exe" utility; **TOWCALC** is compiled using the "large" memory model, and a stack size of 16,384 bytes (hexadecimal 4000). The object code "halodvxx.obj" and the library "haloul.lib" are needed for the Halo graphics routines (the source code for these is not provided).

The include files are:

keydef.h : auxiliary byte values for special keys

plthead.h : function declarations for plotting routines

video.h : various macro definitions

The source files are:

main.c : controls overall program operation

title.c : displays title screen

tug.c : supports "Select Tug" option

tow.c : supports data entry for self-propelled ships

tab.c : displays data base for selection of ship type

dis.c : displays ship data summary; allows editing

dock.c : supports data entry, resistance prediction for drydocks

brg.c : supports data entry, resistance prediction for barges

drg.c : supports resistance prediction for self-propelled ships

pull.c : supports tug evaluation

ext.c : supports estimation of extreme tensions

rp.c : supports "Print Report" option

plt.c : plotting functions

tlib.c : a library of functions used throughout **TOWCALC**

The make file used to compile **TOWCALC** is listed below.

```
main.obj:       main.c
     cl /AL /c main.c

title.obj:      title.c
     cl /AL /c title.c

brg.obj:      brg.c
     cl /AL /c brg.c

dock.obj:       dock.c
     cl /AL /c dock.c

tug.obj:        tug.c
     cl /AL /c tug.c

tow.obj:        tow.c
     cl /AL /c tow.c

tab.obj:        tab.c
     cl /AL /c tab.c

dis.obj:       dis.c
     cl  /AL /c dis.c

drg.obj:        drg.c
     cl /AL /c drg.c

tlib.obj:     tlib.c
     cl /AL /c tlib.c

ext.obj:        ext.c
     cl /AL /c ext.c

pull.obj:       pull.c
     cl /AL /c pull.c

rp.obj:       rp.c
     cl /AL /c rp.c

plt.obj:        plt.c
     cl /AL /c plt.c

towcalc.exe: main.obj title.obj brg.obj dock.obj tug.obj tow.obj tab.obj
dis.obj drg.obj tlib.obj ext.obj pull.obj rp.obj plt.obj halodvxx.obj
     link /ST:0x4000 /NOD main+title+brg+dock+tug+tow+tab+dis+drg+tlib+ext+
pull+rp+plt+halodvxx,towcalc,,llibc+llibfp+libh+em+haloul
```

```
/*****************************************************************************
  File: kyedef.h

    This header file contains definitions for the auxiliary byte values for
  the special keys on an IBM keyboard.
*****************************************************************************/

#define LEFT_ARROW  75
#define RIGHT_ARROW 77
#define UP_ARROW    72
#define DOWN_ARROW  80
#define PAGE_UP     73
#define PAGE_DOWN   81
#define HOME        71
#define END         79
#define ESC         27
#define BKSP         8
#define INSERT      82
#define DELETE      83

#define CTRL_LEFT  115
#define CTRL_RIGHT 116
#define CTRL_END   117
#define CTRL_PGDN  118
#define CTRL_HOME  119
#define CTRL_PGUP  132

#define F1  59
#define F2  60
#define F3  61
#define F4  62
#define F5  63
#define F6  64
#define F7  65
#define F8  66
#define F9  67
#define F10 68

#define SHIFT_F1  84
#define SHIFT_F2  85
#define SHIFT_F3  86
#define SHIFT_F4  87
#define SHIFT_F5  88
#define SHIFT_F6  89
#define SHIFT_F7  90
#define SHIFT_F8  91
#define SHIFT_F9  92
#define SHIFT_F10 93

#define CTRL_F1  94
#define CTRL_F2  95
#define CTRL_F3  96
#define CTRL_F4  97
#define CTRL_F5  98
```

```
#define CTRL_F6  99
#define CTRL_F7  100
#define CTRL_F8  101
#define CTRL_F9  102
#define CTRL_F10 103

#define ALT_F1  104
#define ALT_F2  105
#define ALT_F3  106
#define ALT_F4  107
#define ALT_F5  108
#define ALT_F6  109
#define ALT_F7  110
#define ALT_F8  111
#define ALT_F9  112
#define ALT_F10 113

#define ALT_Q 16
#define ALT_W 17
#define ALT_E 18
#define ALT_R 19
#define ALT_T 20
#define ALT_Y 21
#define ALT_U 22
#define ALT_I 23
#define ALT_O 24
#define ALT_P 25

#define ALT_A 30
#define ALT_S 31
#define ALT_D 32
#define ALT_F 33
#define ALT_G 34
#define ALT_H 35
#define ALT_J 36
#define ALT_K 37
#define ALT_L 38

#define ALT_Z 44
#define ALT_X 45
#define ALT_C 46
#define ALT_V 47
#define ALT_B 48
#define ALT_N 49
#define ALT_M 50
```

```
/*****************************************************************************
  File: plthead.h

    This header file contains declarations for the plotting
  routines in the file plt.c.
*****************************************************************************/

short   ntcl,ix0,iy0,ixl,iyl,npx,nfax,nfay,knumx,knumy,mode,icf,icb,npy,
        ipbx,ipby,ntix,ntiy,nxx,nyy;

float   x0,y0p,xl,yl,xmin,xmax,ymin,ymax,tclx,tcly,scxx,scyy,xrange,
        yrange,xtic,ytic,xtx[100],yty[100],xtl[100],xtu[100],xff,
        yff,ytl[100],ytu[100],x3l[100],x3u[100],y3l[100],y3u[100];

char    cs[6],cll[4],cnul;
```

```
/*****************************************************************************
  File: video.h

    This header file contains various macro definitions used throughout
  TOWCALC.
*****************************************************************************/

#define REV_VID 0x70
#define NORM_VID 7
#define BLINK_REV_VID 0xF0

#define ERROR 0
#define WARN 1
#define BLANK 2

#define NONE 0
#define SINGLE 1
#define DOUBLE 2

#define LOCKED 0
#define TRAILING 1
#define REMOVED 2

#define ARS38 0
#define ARS50 1
#define ATS1 2
#define TATF166 3

#define SHIP 0
#define DOCK 1
#define BARGE 2
```

```
/*****************************************************************************
  File: main.c
  Author: Todd J. Peltzer
  Last update: 29 April 1989

      This file contains the functions which control overall program
  operation.
-----------------------------------------------------------------------------
  Functions:
    main()
    get_options()
    main_menu()
    get_tow_menu()
    initial()
    check_file()
*****************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "stdlib.h"
#include "keydef.h"   /* Define aux byte values for IBM keyboard */
#include "video.h"

static int startrow=2;
static int startcol=20;
static int endrow=22;
static int endcol=60;

static char header[] =
{
  "PROGRAM OPTIONS"
};

static char *options[] =
{
  "1) Select Tug              ",
  "2) Select Tow              ",
  "3) Estimate Dynamic Tension",
  "4) Print Report            ",
  "5) QUIT                    "
};

static char *options1[] =
{
  "1) Enter new data    ",
  "2) Edit existing data",
  "3) Retrieve data file"
};

static char *label[] =
{
  "TOW DATA",
  "Options"
};
```

```
char table[141][81];     /* Table G-2 data                */
char tug[15];            /* tug class                     */
char hull_no[24];        /* hull number entered by user   */
char class[24];          /* class of ship from Table G-2  */
char text1[39];          /* error message text strings    */
char text2[39];          /*   "        "          "       */
int curve;               /* curve number from extreme     */
int flag[4];             /* error checking flag array     */
int type;                /* tow type                      */
float tug_data[5];       /* array to store tug data       */
float tow_data[5];       /* array to store tow data       */
float ship_data[6];      /* array to store Table G-2 data */
float dock_data[7];      /* array to store drydock data   */
float barge_data[8];     /* array to store barge data     */
float resist_dat[5];     /* resistance data               */
float tension;           /* mean towline tension          */
float extr_ten;          /* extreme towline tension       */
float barge_res[6];      /* computed barge parameters     */
float tug_eval[7];       /* tug evaluation results        */
float ext_data[10];      /* extreme tension results       */
float spd_data[3][4];    /* tow speed effects results     */
float lgth_data[3][4];   /* hawser length effects results */

void get_options(),main_menu(),get_tow_menu();
void initial(),check_file();

main()
{
  int i, j;              /* dummy counters */
  char ch;               /* dummy variable */

  for (i=0; i<4; i++)
    flag[i]=0;           /* initialize flags */
  cls();                 /* clear the screen */
  set_video();           /* set video mode */
  initial();             /* check for all necessary data files */
  title();               /* display title screen */
  cursor_off();          /* turn off blinking cursor */

  /* draw background and border */
  draw_window(startrow,startcol,endrow,endcol,DOUBLE,REV_VID);

  get_options(0);        /* display main menu, get user response */

}

/*****************************************************************************
     This function controls the main program options.
*****************************************************************************/
void get_options(start)
int start;
{
  int choice;
  static int extflag;
```

```
/* display main program menu; get user's choice */
main_menu(&choice,start);

if (choice==0)          /* select tug */
{
  /* get tug parameters from user */
  get_tug(tug,tug_data,flag);
  get_options(1);
}
else if (choice==1)     /* select tow */
{
  if (!flag[0])
  {
    sprintf(text1,"A tug has not been selected");
    sprintf(text2,"==> please select a tug <==");
    display_error(ERROR,text1,text2);
    cursor_off();
    get_options(0);
  }
  get_tow_menu(&type);
}
else if (choice==2)     /* estimate dynamic tension */
{
  if (!flag[0] && !flag[1])
  {
    sprintf(text1,"Tug and tow have not been selected");
    sprintf(text2,"==> please select a tug and a tow <==");
    display_error(ERROR,text1,text2);
    cursor_off();
    get_options(0);
  }
  else if (!flag[0])
  {
    sprintf(text1,"A tug has not been selected");
    sprintf(text2,"==> please select a tug <==");
    display_error(ERROR,text1,text2);
    cursor_off();
    get_options(0);
  }
  else if (!flag[1])
  {
    sprintf(text1,"A tow has not been selected");
    sprintf(text2,"==> please select a tow <==");
    display_error(ERROR,text1,text2);
    cursor_off();
    get_options(1);
  }
  extreme(type,&tension,&extr_ten,&curve,&extflag);
  get_ext_opt(extflag,type,curve,tension,extr_ten,0);
  flag[3]=1;
  get_options(2);

}
```

```
  else if (choice==3)     /* print reports */
  {
    if (!flag[0] && !flag[1])
    {
      sprintf(text1,"Tug and tow have not been selected");
      sprintf(text2,"==> please select a tug and a tow <==");
      display_error(ERROR,text1,text2);
      cursor_off();
      get_options(0);
    }
    else if (!flag[0])
    {
      sprintf(text1,"A tug has not been selected");
      sprintf(text2,"==> please select a tug <==");
      display_error(ERROR,text1,text2);
      cursor_off();
      get_options(0);
    }
    else if (!flag[1])
    {
      sprintf(text1,"A tow has not been selected");
      sprintf(text2,"==> please select a tow <==");
      display_error(ERROR,text1,text2);
      cursor_off();
      get_options(1);
    }
    else if (!flag[3])
    {
      sprintf(text1,"Dynamic tension has not been estimated");
      sprintf(text2,"==> please select Dynamic Tension <==");
      display_error(ERROR,text1,text2);
      cursor_off();
      get_options(2);
    }
    report();
    cursor_off();
    get_options(4);
  }
  else if (choice==4 || choice<0)     /* quit */
  {
    cls();
    cursor_on();
    exit(0);
  }
}

/*****************************************************************************
     This function displays the main program menu and allows the user to
  select an option using the cursor keys.
*****************************************************************************/
void main_menu(choice,hilite)
int *choice;
int hilite;
{
```

```
  int i,start,end;

  /* clear background */
  clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,header,REV_VID);

  /* write options */
  start = (endcol-startcol-strlen(options[3]))/2 + startcol;
  for (i=0; i<6; i++)
  {
    write_string(startrow+7+i,start,options[i],REV_VID);
  }

  /* display options with popup menu; get response */
  *choice = popup(options,"12345",5,startrow+6,start-2,NONE,REV_VID,hilite);

}


/*************************************************************************
     This function displays the tow menu and allows the user to select an
  option using the cursor keys.
*************************************************************************/
void get_tow_menu(shiptype)
int *shiptype;
{
  int i,start,choice,status,docktype;
  int row1=startrow+8;          /* boundaries of menu */
  int row2=startrow+15;         /*          "         */
  int col1=startcol+5;          /*          "         */
  int col2=startcol+35;         /*          "         */
  FILE *in;                     /* pointer to file  */
  unsigned char *p;             /* buffer for video */

  static char header[] =
  {
    "Select Type"
  };

  static char *type[] =
  {
    "1) Self-propelled ships",
    "2) Floating drydocks   ",
    "3) Barges              "
  };

  /* clear background */
  clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,label[0],REV_VID);
```

```
/* select enter new data, edit current data, or retrieve file */
start = (endcol-startcol-strlen(label[1]))/2 + startcol;
write_string(startrow+8,start,label[1],REV_VID);

start = (endcol-startcol-strlen(options1[0]))/2 + startcol;
choice = popup(options1,"123",3,startrow+10,start-2,NONE,REV_VID,0);

if (choice==0)                /* enter new data */
{
  clear(startrow+5,startcol+1,endrow-3,endcol-1,REV_VID);
  flag[1]=0;
}
else if (choice==1)    /* edit existing data */
{
  if (!flag[1])         /* tow data does not exist */
  {
    sprintf(text1,"Tow data has not been set; please");
    sprintf(text2,"enter new data or get from file");
    display_error(ERROR,text1,text2);
    cursor_off();
    get_tow_menu(shiptype);
    return;
  }
  flag[2]=0;
}
else if (choice==2)    /* retrieve data file */
{
  flag[1]=0;
  flag[2]=1;
}
else if (choice<0)     /* ESCAPE key pressed */
{
  get_options(1);
  return;
}

if (!flag[1])     /* tow data not set */
{
  /* display options with popup menu; get response */
  clear(startrow+5,startcol+1,endrow-1,endcol-1,REV_VID);
  start = (endcol-startcol-strlen(header))/2 + startcol;
  write_string(startrow+8,start,header,REV_VID);
  start = (endcol-startcol-strlen(type[0]))/2 + startcol;
  *shiptype = popup(type,"123",3,startrow+10,start-2,NONE,REV_VID,0);
}                  /********/
if (*shiptype==0)  /* ship */
{                  /********/
  /* get ship tow parameters from user */
  status=get_tow_data(flag,hull_no,class,tow_data,ship_data);
  if (status<0)
  {
    cursor_off();
    get_tow_menu(shiptype);
    return;
```

```
  }

  if (!flag[1] && !flag[2])   /* ship data not set */
  {
    /* read string data from file */
    in = fopen("tab_str.dat","r");
    read_table(in,table,80);  /* 80 is length of cursor highlight */
                              /*         (full screen)            */
    fclose(in);

    /* display Table G-2, get user's choice, and read data */
    display_tab_g2(table,ship_data,hull_no,class);

    cls();
  }

  /* display data from selected ship class */
  display_data(hull_no,class,tow_data,ship_data);

  /* compute mean towline tension, display results & summary */
  mean_tension(tug_data,tow_data,ship_data,hull_no,class,&tension);

  /* evaluate tug */
  tugpull(*shiptype,&tension);

  flag[1]=1;
  get_options(2);
  return;
}                       /***********/
else if (*shiptype==1)  /* drydock */
{                       /***********/
  /* get drydock type */
  status=get_dock(flag,&docktype,dock_data,hull_no,tow_data);
  if (status<0)
  {
    cursor_off();
    get_tow_menu(shiptype);
    return;
  }

  if(!flag[1] && !flag[2])         /* tow data not set */
  {
    /* get hull condition */
    get_hull_cond(tow_data);

    /* get tow speed and wind speed */
    get_dock_towdata(tow_data);

    /* estimate displacement for extreme tension calculations */
    est_disp(0,dock_data,tow_data);
  }

  /* display summary; give edit option */
  dock_summary(&docktype,dock_data,tow_data,hull_no);
```

```
    /* compute resistance and display results */
    get_dock_resist(hull_no,tug_data,dock_data,tow_data,&tension);

    /* evaluate tug */
    tugpull(*shiptype,&tension);

    flag[1]=1;
    get_options(2);
    return;
  }                          /*********/
  else if (*shiptype==2)  /* barge */
  {                          /*********/
    /* enter barge data & compute resistance */
    status=get_barge_resist(flag,hull_no,tug_data,tow_data,
                barge_data,&tension);

    if (status<0)
    {
      cursor_off();
      get_tow_menu(shiptype);
      return;
    }

    /* evaluate tug */
    tugpull(*shiptype,&tension);

    flag[1]=1;
    get_options(2);
    return;
  }
  else
  {
    get_options(1);
    return;
  }
}

/*****************************************************************
  This function checks for the presence of all necessary data
  files by repeated calls to check_file() before continuing
  with program execution.
*****************************************************************/
void initial()
{
  FILE *in;
  char fname[13];
  char device[13];

  sprintf(fname,"DEVICE.DAT");
  check_file(fname);

  in=fopen(fname,"r");
  fgets(device,13,in);    /* read graphics driver filename */
```

```
  fclose(in);

  check_file(device);

  sprintf(fname,"HALO104.FNT");
  check_file(fname);

  sprintf(fname,"HALO105.FNT");
  check_file(fname);

  sprintf(fname,"HALO106.FNT");
  check_file(fname);

  sprintf(fname,"HALO201.FNT");
  check_file(fname);

  sprintf(fname,"TAB_STR.DAT");
  check_file(fname);

  sprintf(fname,"TABLE.DAT");
  check_file(fname);

  sprintf(fname,"DOCK_STR.DAT");
  check_file(fname);

  sprintf(fname,"DRYDOCK.DAT");
  check_file(fname);

  sprintf(fname,"RS_WVHT.DAT");
  check_file(fname);

  sprintf(fname,"RH_DISP.DAT");
  check_file(fname);

  sprintf(fname,"TUGPULL.DAT");
  check_file(fname);

  sprintf(fname,"CURVE.TAB");
  check_file(fname);
)


/*****************************************************************
  This function checks for the presence of a given file. If not
  present, program execution is terminated with error code 1.
*****************************************************************/
void check_file(fname)
char *fname;
{
  FILE *in;
  int i;
  char text1[37],text2[37];

  if ( (in=fopen(fname,"r"))==NULL)
```

```
  {
    sprintf(text1,"Missing file %s",fname);
    sprintf(text2,"==> please reload file from disk <==");
    display_error(ERROR,text1,text2);
    exit(1);
  }
  fclose(in);
}
```

```
/*************************************************************************
  File: title.c
  Author: Todd J. Peltzer
  Last update: 30 April 1989

       This file contains the function title() which uses Halo 88 (tm)
  graphics to display the opening screen of "TOWCALC".
-------------------------------------------------------------------------
  Functions:
    title()
*************************************************************************/
#include "stdio.h"

void title()
{
    FILE *in;
    char device[13];            /* variable for device driver name */

    union inkey {
      char ch[2];
      int i;
    } c;

    static char font[] = "HALO201.FNT";              /* Font file name */
    static char font2[] = "HALO104.FNT";             /* Font file name */
    static char font3[] = "HALO105.FNT";             /* Font file name */
    static char font4[] = "HALO106.FNT";             /* Font file name */

    static char string[]   = "SUPSALV";                  /* Text string */
    static char string2[] = "TOWING CALCULATIONS";       /*     "       */
    static char string3[] = "A COMPUTATIONAL TOOL FOR";  /*     "       */
    static char string4[] = "OPEN OCEAN TOWING";         /*     "       */
    static char string5[] = "Press any key to continue"; /*     "       */

    float height = 100.0;                          /* Stroke text height */
    float asp = 1.0;                         /* Stroke text aspect ratio */
    int path = 0;                                 /* Stroke text path        */
    float iheight;                      /* Inquired height of text string */
    float width;                         /* Inquired width of text String */
    float offse;                                /* Offset for descenders */
    float tx,ty;              /* X and y coordinate text cursor location */
    int maxcolor;                              /* Maximum color variable */
    int color;                       /* Color variable for screen clear */
    int style = 1;               /* Initialize the style for hatchstyle */

    float x1,y1,x2,y2;                       /* World coordinates values */
    float vx1,vy1,vx2,vy2;                   /* Viewport coordinates     */
    float x,y;                               /* Graphics cursor position */
    int i,j,n;                                     /* Counter Variables */
    int mode;                                          /* Graphics mode */

    int border,back;           /* Viewport background and border values */

    in = fopen("device.dat","r");
```

```
    n=fscanf(in,"%s\n",device);          /* read device driver from file */
    n=fscanf(in,"%d\n",&mode);           /* read graphics mode from file */
    fclose(in);

    setdev(device);                    /* Initialize the graphics device */

    initgraphics( &mode );
    inqcrange(&maxcolor);                          /* Inquire maximum color */

    x1 = 0.0;
    y1 = 0.0;
    x2 = 1000.0;
    y2 = 1000.0;
    setworld(&x1,&y1,&x2,&y2);

    vx1 = 0.1;
    vy1 = 0.1;
    vx2 = 0.9;
    vy2 = 0.9;
    border = maxcolor;
    back = 0;
    setviewport(&vx1,&vy1,&vx2,&vy2,&border,&back);

/*************************************************************************
*           Display the first text string                                *
*************************************************************************/
    setfont(font);
    setstclr(&maxcolor,&maxcolor);
    setstext(&height,&asp,&path);
    inqstsize(string,&iheight,&width,&offse);
    tx = (x2-width)/2;
    ty = 700.0;
    movtcurabs(&tx,&ty);
    stext(string);

/*************************************************************************
*               Display the second text string                           *
*************************************************************************/
    style = 1;
    setfont(font2);
    setstclr(&maxcolor,&maxcolor);
    setstext(&height,&asp,&path);
    inqstsize(string2,&iheight,&width,&offse);
    tx = (x2-width)/2;
    ty = 500.0;
    movtcurabs(&tx,&ty);
    stext(string2);

/*************************************************************************
*               Display the third text string                            *
*************************************************************************/
    style = 1;
    height = 50.0;
    setfont(font3);
```

```
    setstclr(&maxcolor,&maxcolor);
    setstext(&height,&asp,&path);
    inqstsize(string3,&iheight,&width,&offse);
    tx = (x2-width)/2;
    ty = 350.0;
    movtcurabs(&tx,&ty);
    stext(string3);

/*************************************************************************
*               Display the fourth text string                           *
*************************************************************************/
    inqstsize(string4,&iheight,&width,&offse);
    tx = (x2-width)/2;
    ty = 250.0;
    movtcurabs(&tx,&ty);
    stext(string4);

/*************************************************************************
*               Display the fifth text string                            *
*************************************************************************/
    style = 1;
    height = 40.0;
    setfont(font4);
    setstclr(&maxcolor,&maxcolor);
    setstext(&height,&asp,&path);
    inqstsize(string5,&iheight,&width,&offse);
    tx = (x2-width)/2;
    ty = 75.0;
    movtcurabs(&tx,&ty);
    stext(string5);

    deltcur();

    /* wait for keystroke, then exit */
    while (1)
    {
      c.i = bioskey(0);        /* read the key */

      if (c.ch[0])             /* key is a normal key */
      {
        break;
      }
      else                     /* key is a special key */
      {
        break;
      }
    }

    closegraphics();                              /* Close graphics */
}
```

```
/*****************************************************************************
  File: tug.c
  Author: Todd J. Peltzer
  Last update: 30 April 1989

       This file contains the functions which support the "Select Tug"
  option in the main program.
-----------------------------------------------------------------------------
  Functions:
    get_tug()
    get_tug_data()
    get_tug_file()
    get_fname()
    save_tug_file()
*****************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "keydef.h"
#include "video.h"

static char *menu[] =
{
  "1) Enter new data    ",
  "2) Edit existing data",
  "3) Retrieve data file"
};

static char *label1[] =
{
  "Options",
  "Retrieve File",
  "Enter name of tug file",
  "to retrieve (8 char max): "
};

static char *label[] =
{
  "TUG DATA",
  "Class:",
  "Hawser",
  "  diameter:",
  "  scope:",
  "Chain pendant:",
  "  size:",
  "  scope:"
};

static char *units[] =
{
  "in",
  "ft",
  "in",
  "ft"
};
```

```
static char *footer[] =
{
  "Please enter data.",
  "Is all data correct? (yes/no): ",
  "F1 Class F2 Wire scope",
  "F3 Chain size F4 Chain scope",
  " Press INS to continue ",
  "  Save data to file? (yes/no): ",
  " Enter tug file name: "
};

char *tug_menu[]= {
   "ARS 38   ",
   "ARS 50   ",
   "ATS 1    ",
   "T-ATF 166"
};

static char choice[4];
static char fname[9];

typedef struct
{
  int startcol;
  int endcol;
} data_box;

static data_box input[] =
  {
        {30,45},        /* tug            */
        {35,41},        /* hawser dia     */
        {35,41},        /* hawser scope   */
        {35,41},        /* chain size     */
        {35,41},        /* chain scope    */
        {54,58},        /* yes/no choice  */
        {49,58}         /* tug file name  */
};

int get_tug_file();
void get_tug_data(),get_fname();
void save_tug_file();

/*****************************************************************************
     This function prompts the user for data concerning the tug. A popup
  menu is used to select the tug class; hawser diameter is selected auto-
  matically based on this choice. Hawser scope, chain size, and chain scope
  are all input by the user.
*****************************************************************************/
void get_tug(tug,tug_data,flag)
char *tug;
float *tug_data;
int *flag;
{
```

```
FILE *in, *out;
int startrow=2;
int startcol=20;
int endrow=22;
int endcol=60;

int row,col,i,start,start1,key;
int status,choice1;
char string[25];
char fname[13];
char text1[39], text2[39];    /* error message text */

if (!flag[0])                 /* tug data does not exist */
{
  /* initialize data variables to zero */
  tug[0] = NULL;
  for (i=0; i<5; i++)
    tug_data[i] = 0.0;
}

/* clear background  */
clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

/* write header */
write_header(startrow,startcol,endcol,label[0],REV_VID);

/* select retrieve file or enter new data */
start = (endcol-startcol-strlen(label1[0]))/2 + startcol;
write_string(startrow+8,start,label1[0],REV_VID);

start = (endcol-startcol-strlen(menu[0]))/2 + startcol;
choice1 = popup(menu,"123",3,startrow+10,start-2,NONE,REV_VID,0);

if (choice1==0)          /* enter new data */
{
  clear(startrow+5,startcol+1,endrow-3,endcol-1,REV_VID);
}
else if (choice1==1)     /* edit existing data */
{
  if (!flag[0])
  {
    sprintf(text1,"Tug data has not been set; please");
    sprintf(text2,"enter new data or get from file");
    display_error(ERROR,text1,text2);
    cursor_off();
    get_tug(tug,tug_data,flag);
    return;
  }
}
else if (choice1==2)     /* retrieve data from file */
{
  status=get_tug_file(startrow,startcol,endrow,endcol,tug,tug_data,flag);
  if (status<0)
  {
```

```
    cursor_off();
    get_tug(tug,tug_data,flag);
    return;
  }
}
else if (choice1<0)     /* ESCAPE key pressed */
{
  get_options(0);
  return;
}
clear(startrow+5,startcol+1,endrow-3,endcol-1,REV_VID);

/* write data labels */
write_string(startrow+5,startcol+3,label[1],REV_VID);
write_string(startrow+7,startcol+3,label[2],REV_VID);
write_string(startrow+8,startcol+3,label[3],REV_VID);
write_string(startrow+10,startcol+3,label[4],REV_VID);
write_string(startrow+12,startcol+3,label[5],REV_VID);
write_string(startrow+13,startcol+3,label[6],REV_VID);
write_string(startrow+15,startcol+3,label[7],REV_VID);

/* write units */
write_string(startrow+8,startcol+2+strlen(label[3])+2+7,units[0],REV_VID);
write_string(startrow+10,startcol+2+strlen(label[3])+2+7,units[1],REV_VID);
write_string(startrow+13,startcol+2+strlen(label[3])+2+7,units[2],REV_VID);
write_string(startrow+15,startcol+2+strlen(label[3])+2+7,units[3],REV_VID);

if (choice1==1 || choice1==2)
{
  /* display data */
  /* tug class */
  write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);

  /* hawser diameter */
  sprintf(string,"%6.2f",tug_data[1]);
  write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);

  /* hawser scope */
  sprintf(string,"%6.0f",tug_data[2]);
  write_string(startrow+10,startcol+2+strlen(label[3])+2,string,REV_VID);

  /* chain pendant size */
  sprintf(string,"%6.2f",tug_data[3]);
  write_string(startrow+13,startcol+2+strlen(label[3])+2,string,REV_VID);

  /* chain pendant scope */
  sprintf(string,"%6.0f",tug_data[4]);
  write_string(startrow+15,startcol+2+strlen(label[3])+2,string,REV_VID);
}
else if (choice1==0)
{
  /* write footer */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  write_string(endrow-3,start,footer[0],REV_VID);
```

```
/* create normal video boxes for data entry */
vid_box(startrow+10,startcol+2+strlen(label[3])+2,6);
vid_box(startrow+13,startcol+2+strlen(label[3])+2,6);
vid_box(startrow+15,startcol+2+strlen(label[3])+2,6);

/*********************************************************/
/*                     get user input                    */
/*********************************************************/
/* get tug class */
cursor_off();
tug_data[0] = popup(tug_menu,"",4,startrow+4,startcol+15,SINGLE,REV_VID,0);
if (tug_data[0] == ARS38)
{
  sprintf(tug,"ARS 38");
  write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);
  tug_data[1] = 2.0;
  sprintf(string,"%6.2f",tug_data[1]);
  write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);
}
else if (tug_data[0] == ARS50)
{
  sprintf(tug,"ARS 50");
  write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);
  tug_data[1] = 2.25;
  sprintf(string,"%6.2f",tug_data[1]);
  write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);
}
else if (tug_data[0] == ATS1)
{
  sprintf(tug,"ATS 1");
  write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);
  tug_data[1] = 2.25;
  sprintf(string,"%6.2f",tug_data[1]);
  write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);
}
else if (tug_data[0] == TATF166)
{
  sprintf(tug,"T-ATF 166");
  write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);
  tug_data[1] = 2.25;
  sprintf(string,"%6.2f",tug_data[1]);
  write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);
}
else
{
  get_tug(tug,tug_data,flag);
  return ;
}

/* get hawser scope */
get_tug_data(2,startrow+10,startcol+2+strlen(label[3])+2,tug,tug_data);
for (i=0; i<6; i++)
  write_char(startrow+10,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
```

```
  sprintf(string,"%6.0f",tug_data[2]);
  write_string(startrow+10,startcol+2+strlen(label[3])+2,string,REV_VID);

  /* get chain pendant size */
  get_tug_data(3,startrow+13,startcol+2+strlen(label[3])+2,tug,tug_data);
  for (i=0; i<6; i++)
    write_char(startrow+13,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
  sprintf(string,"%6.2f",tug_data[3]);
  write_string(startrow+13,startcol+2+strlen(label[3])+2,string,REV_VID);

  /* get chain pendant scope */
  get_tug_data(4,startrow+15,startcol+2+strlen(label[3])+2,tug,tug_data);
  for (i=0; i<6; i++)
    write_char(startrow+15,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
  sprintf(string,"%6.0f",tug_data[4]);
  write_string(startrow+15,startcol+2+strlen(label[3])+2,string,REV_VID);


  /* erase footer */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  for (i=0; i<strlen(footer[0]); i++)
    write_char(endrow-3,start+i,' ',REV_VID);
}

/*****************************************************************/
/*    prompt user for confirmation of data; give edit option     */
/*****************************************************************/
start = startcol + 3;
write_string(endrow-3,start,footer[1],REV_VID);
vid_box(endrow-3,start+strlen(footer[1]),4);
get_tug_data(5,endrow-3,start+strlen(footer[1]),tug,tug_data);

/* test if data correct */
if ( choice[0]!='y' && choice[0]!='Y' && choice[0]!=NULL )
{
  /* erase previous message */
  start = startcol + 3;
  for (i=0; i<strlen(footer[1])+4; i++)
    write_char(endrow-3,start+i,' ',REV_VID);

  /* write quit message */
  start=(endcol-startcol-strlen(footer[4]))/2 + startcol;
  write_string(endrow,start,footer[4],REV_VID);

  /* write first line of edit "menu" */
  start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
  write_string(endrow-3,start,footer[2],REV_VID);

  /* write function key indicators in NORM_VID */
  for(i=0; i<2; i++)
  {
    write_char(endrow-3,start+i,footer[2][i],NORM_VID);
    write_char(endrow-3,start+9+i,footer[2][i+9],NORM_VID);
  }
```

```
/* write second line of edit "menu" */
start1=(endcol-startcol-strlen(footer[3]))/2 + startcol;
write_string(endrow-2,start1,footer[3],REV_VID);

/* write function key indicators in NORM_VID */
for(i=0; i<2; i++)
{
  write_char(endrow-2,start1+i,footer[3][i],NORM_VID);
  write_char(endrow-2,start1+14+i,footer[3][i+14],NORM_VID);
}

/*************************************************************/
/*                   edit data                               */
/*************************************************************/
/* move cursor to F1 highlight */
cursor_on();
goto_xy(endrow-3,start);

/* get user's choice */
while (1)
{
  key=get_special();
                  /**************/
  if (key==F1)    /* tug class  */
  {               /**************/
    cursor_off;
    tug_data[0] = popup(tug_menu,"",4,startrow+4,startcol+15,SINGLE,REV_VID,0);
    if (tug_data[0] == ARS38)
    {
      for (i=0; i<10; i++)
        write_char(startrow+5,startcol+2+strlen(label[1])+2+i,' ',REV_VID);
      sprintf(tug,"ARS 38");
      write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);
      tug_data[1] = 2.0;
      sprintf(string,"%6.2f",tug_data[1]);
      write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);

      /* check data for consistency */
      if (tug_data[2]>2100)
      {
        sprintf(text1,"Maximum hawser length for this class");
        sprintf(text2,"is 2100 ft--please enter new length");
        display_error(ERROR,text1,text2);
        vid_box(startrow+10,startcol+2+strlen(label[3])+2,6);
        get_tug_data(2,startrow+10,startcol+2+strlen(label[3])+2,tug,tug_data);
        for (i=0; i<6; i++)
          write_char(startrow+10,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
        sprintf(string,"%6.0f",tug_data[2]);
        write_string(startrow+10,startcol+2+strlen(label[3])+2,string,REV_VID);
      }
    }
    else if (tug_data[0] == ARS50)
    {
```

```
  for (i=0; i<10; i++)
    write_char(startrow+5,startcol+2+strlen(label[1])+2+i,' ',REV_VID);
  sprintf(tug,"ARS 50");
  write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);
  tug_data[1] = 2.25;
  sprintf(string,"%6.2f",tug_data[1]);
  write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);

  /* check data for consistency */
  if (tug_data[2]>3000)
  {
    sprintf(text1,"Maximum hawser length for this class");
    sprintf(text2,"is 3000 ft--please enter new length");
    display_error(ERROR,text1,text2);
    vid_box(startrow+10,startcol+2+strlen(label[3])+2,6);
    get_tug_data(2,startrow+10,startcol+2+strlen(label[3])+2,tug,tug_data);
    for (i=0; i<6; i++)
      write_char(startrow+10,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
    sprintf(string,"%6.0f",tug_data[2]);
    write_string(startrow+10,startcol+2+strlen(label[3])+2,string,REV_VID);
  }
}
else if (tug_data[0] == ATS1)
{
  for (i=0; i<10; i++)
    write_char(startrow+5,startcol+2+strlen(label[1])+2+i,' ',REV_VID);
  sprintf(tug,"ATS 1");
  write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);
  tug_data[1] = 2.25;
  sprintf(string,"%6.2f",tug_data[1]);
  write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);

  /* check data for consistency */
  if (tug_data[2]>3000)
  {
    sprintf(text1,"Maximum hawser length for this class");
    sprintf(text2,"is 3000 ft--please enter new length");
    display_error(ERROR,text1,text2);
    vid_box(startrow+10,startcol+2+strlen(label[3])+2,6);
    get_tug_data(2,startrow+10,startcol+2+strlen(label[3])+2,tug,tug_data);
    for (i=0; i<6; i++)
      write_char(startrow+10,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
    sprintf(string,"%6.0f",tug_data[2]);
    write_string(startrow+10,startcol+2+strlen(label[3])+2,string,REV_VID);
  }
}
else if (tug_data[0] == TATF166)
{
  for (i=0; i<10; i++)
    write_char(startrow+5,startcol+2+strlen(label[1])+2+i,' ',REV_VID);
  sprintf(tug,"T-ATF 166");
  write_string(startrow+5,startcol+2+strlen(label[1])+2,tug,REV_VID);
  tug_data[1] = 2.25;
  sprintf(string,"%6.2f",tug_data[1]);
```

```
    write_string(startrow+8,startcol+2+strlen(label[3])+2,string,REV_VID);

    /* check data for consistency */
    if (tug_data[2]>2500)
    {
      sprintf(text1,"Maximum hawser length for this class");
      sprintf(text2,"is 2500 ft--please enter new length");
      display_error(ERROR,text1,text2);
      vid_box(startrow+10,startcol+2+strlen(label[3])+2,6);
      get_tug_data(2,startrow+10,startcol+2+strlen(label[3])+2,tug,tug_data);
      for (i=0; i<6; i++)
        write_char(startrow+10,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
      sprintf(string,"%6.0f",tug_data[2]);
      write_string(startrow+10,startcol+2+strlen(label[3])+2,string,REV_VID);
    }
  }
  cursor_on();
  goto_xy(endrow-3,start);
}                    /***************/
else if (key==F2)    /* wire scope */
{                    /***************/
  vid_box(startrow+10,startcol+2+strlen(label[3])+2,6);
  get_tug_data(2,startrow+10,startcol+2+strlen(label[3])+2,tug,tug_data);
  for (i=0; i<6; i++)
    write_char(startrow+10,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
  sprintf(string,"%6.0f",tug_data[2]);
  write_string(startrow+10,startcol+2+strlen(label[3])+2,string,REV_VID);
  cursor_on();
  goto_xy(endrow-3,start+9);
}                   /***************/
else if (key==F3)   /* chain size */
{                   /***************/
  vid_box(startrow+13,startcol+2+strlen(label[3])+2,6);
  get_tug_data(3,startrow+13,startcol+2+strlen(label[3])+2,tug,tug_data);
  for (i=0; i<6; i++)
    write_char(startrow+13,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
  sprintf(string,"%6.2f",tug_data[3]);
  write_string(startrow+13,startcol+2+strlen(label[3])+2,string,REV_VID);
  cursor_on();
  goto_xy(endrow-2,start1);
}                    /****************/
else if (key==F4)    /* chain scope */
{                    /****************/
  vid_box(startrow+15,startcol+2+strlen(label[3])+2,6);
  get_tug_data(4,startrow+15,startcol+2+strlen(label[3])+2,tug,tug_data);
  for (i=0; i<6; i++)
    write_char(startrow+15,startcol+2+strlen(label[3])+2+i,' ',REV_VID);
  sprintf(string,"%6.0f",tug_data[4]);
  write_string(startrow+15,startcol+2+strlen(label[3])+2,string,REV_VID);
  cursor_on();
  goto_xy(endrow-2,start1+14);
}                       /*******************/
else if (key==INSERT)   /* quit edit mode */
{                       /*******************/
```

```
        /* erase menu */
        for (i=0; i<strlen(footer[2]); i++)
          write_char(endrow-3,start+i,' ',REV_VID);
        for (i=0; i<strlen(footer[3]); i++)
          write_char(endrow-2,start1+i,' ',REV_VID);

        /* erase message */
        start = (endcol-startcol-strlen(footer[4]))/2 + startcol;
        for (i=0; i<strlen(footer[4]); i++)
          write_char(endrow,start+i,205,REV_VID);
        break;
      }
    )
  cursor_off();
  }
  else      /* data is correct; no editing necessary */
  {
    /* erase previous message */
    start = startcol + 3;
    for (i=0; i<strlen(footer[1])+4; i++)
      write_char(endrow-3,start+i,' ',REV_VID);
  )
  /* prompt to save data to file */
  start = startcol + 3;
  write_string(endrow-3,start,footer[5],REV_VID);
  vid_box(endrow-3,start+strlen(footer[5]),4);
  get_tug_data(5,endrow-3,start+strlen(footer[5]),tug,tug_data);

  if ( choice[0]!='n' && choice[0]!='N' )    /* save data */
  {
    /* erase previous message */
    start = startcol + 3;
    for (i=0; i<strlen(footer[1])+4; i++)
      write_char(endrow-3,start+i,' ',REV_VID);

    save_tug_file(startrow,startcol,endrow,endcol,tug,tug_data);
  }
  flag[0]=1;            /* set flag for presence of tug data */
}

/*****************************************************************************
     This function gets the user's input for the data requested in the
  function get_tug(). Uses screen_getstring() for screen I/O. Some data
  checking is performed for each data type.
*****************************************************************************/
void get_tug_data(i,row,col,tug,data)
int i;
int row;
int col;
char *tug;
float *data;
{
  int nbr;                  /* required arg for stofa()       */
  int status;               /* takes return value for stofa() */
```

```
char string[24];            /* input string                    */
char text1[39], text2[39]; /* error message text              */
int j, k;                   /* counters                        */
int length;
char ch;

cursor_on();                /* turn on cursor                  */
goto_xy(row,col);           /* move cursor to start of box     */

/* get user input */
screen_getstrg(i,row,col,string,NORM_VID,input);

if (i>1 && i<5)
{
  /* check for valid numeric input */
  for (k=0; k<strlen(string); k++)
  (
    if ( string[k]=='.' ) ;
    else if (string[k]<'0' || string[k]>'9')
    (
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
      vid_box(row,col,6);
      get_tug_data(i,row,col,tug,data);
      return ;
    )
  )
  /* if input string contains valid numbers only, convert string to float */
  status = stofa(string,&data[i],&nbr,1);
}
else   /* i==5 */
{
  strcpy(choice,string);
)

/*************************************************/
/*            check data for consistency         */
/*************************************************/
if (i==2)    /* check hawser scope */
{
  if (data[i]<=0.0)
  (
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,6);
    get_tug_data(i,row,col,tug,data);
    return ;
  )
  if (data[i]<1000)
  {
    sprintf(text1,"No extreme tension data for scopes");
    sprintf(text2,"less than 1000' -- please try again");
    display_error(ERROR,text1,text2);
    vid_box(row,col,6);
```

```
    get_tug_data(i,row,col,tug,data);
    return;
  )
  if (data[0]==ARS38 && data[i]>2100)
  (
    sprintf(text1,"Maximum hawser length for this class");
    sprintf(text2,"is 2100 ft--please try again");
    display_error(ERROR,text1,text2);
    vid_box(row,col,6);
    get_tug_data(i,row,col,tug,data);
    return;
  }
  if ( (data[0]==ARS50 || data[0]==ATS1) && data[i]>3000)
  {
    sprintf(text1,"Maximum hawser length for this class");
    sprintf(text2,"is 3000 ft--please try again");
    display_error(ERROR,text1,text2);
    vid_box(row,col,6);
    get_tug_data(i,row,col,tug,data);
    return;
  }
  if (data[0]==TATF166 && data[i]>2500)
  (
    sprintf(text1,"Maximum hawser length for this class");
    sprintf(text2,"is 2500 ft--please try again");
    display_error(ERROR,text1,text2);
    vid_box(row,col,6);
    get_tug_data(i,row,col,tug,data);
    return;
  )
}
if (i==3)    /* check chain size */
{
  if (data[i]<0 || data[i]>20.0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,6);
    get_tug_data(i,row,col,tug,data);
    return ;
  }
)
if (i==4)    /* check chain scope */
{
  if (data[i]<0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,6);
    get_tug_data(i,row,col,tug,data);
    return ;
  }
  else if (data[i]>500.0)
  {
```

```
      sprintf(text1,"Scope of chain is unusually large");
      sprintf(text2,"=>please check this entry<=");
      display_error(WARN,text1,text2);
      vid_box(row,col,6);
      get_tug_data(i,row,col,tug,data);
      return ;
    }
  }

  cursor_off();
  goto_xy(0,0);
}

/*****************************************************************
  This function retrieves tug data from a user specified file.
*****************************************************************/
get_tug_file(startrow,startcol,endrow,endcol,tug,tug_data,flag)
int startrow,startcol;
int endrow,endcol;
char *tug;
float *tug_data;
int *flag;
{
  FILE *in;
  char fname[13];
  char inline[81];
  char text1[39],text2[39];
  int i,j,status,nbr;
  float array[2];
  int start;

  start = (endcol-startcol-strlen(label1[1]))/2 + startcol;
  write_string(startrow+8,start,label1[1],REV_VID);
  for (i=0; i<2; i++)     /* write prompt */
  {
    start = startcol+3;
    write_string(startrow+10+i,start,label1[i+2],REV_VID);
  }
  vid_box(startrow+11,start+strlen(label1[3]),9);
  sprintf(text1,"Type 'Q' to quit");
  start = (endcol-startcol-strlen(text1))/2 + startcol;
  write_string(endrow-3,start,text1,REV_VID);
  get_fname(6,0,startrow+11,startcol+3+strlen(label1[3]),fname);

  /* check for quit option */
  if (fname[0]=='Q')
  {
    cursor_off();
    goto_xy(0,0);
    return (-1);
  }

  in=fopen(fname,"r");
  if (in==NULL)
```

```
{
  sprintf(text1,"Can't open file %s",fname);
  display_error(ERROR,text1," ");
  clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
  get_tug_file(startrow,startcol,endrow,endcol,tug,tug_data,flag);
  return (0);
}
else
{
  /* read data from file */
  fgets(inline,81,in);
  strstrip(inline);
  if (!(strncmp(inline,"! Tug data file",15)))
  {
    fgets(inline,81,in);
    fgets(inline,81,in);
    strncpy(tug,inline,15);
    strstrip(tug);

    /* test for tug type, assign value to tug_data[0] */
    if (!(strncmp(tug,"ARS 38",6)))
      tug_data[0]=0;
    else if (!(strncmp(tug,"ARS 50",6)))
      tug_data[0]=1;
    else if (!(strncmp(tug,"ATS 1",5)))
      tug_data[0]=2;
    else if (!(strncmp(tug,"T-ATF 166",9)))
      tug_data[0]=3;
    else
    {
      sprintf(text1,"Tug type in file is not in data base");
      sprintf(text2,"==> please try again <==");
      display_error(ERROR,text1,text2);
      get_tug_file(startrow,startcol,endrow,endcol,tug,tug_data,flag);
      return (0);
    }
    /* read remaining data */
    for (i=0; i<4; i++)
    {
      fgets(inline,81,in);
      fgets(inline,81,in);
      status=stofa(inline,array,&nbr,1);
      tug_data[i+1]=array[0];
    }
  fclose(in);
  }
  else
  {
    sprintf(text1,"%s is not a tug data file!",fname);
    display_error(ERROR,text1," ");
    clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
    get_tug_file(startrow,startcol,endrow,endcol,tug,tug_data,flag);
    return (1);
  }
```

```
  }
  return(0);
}


/*********************************************************************
  This function gets the filename from the user.
*********************************************************************/
void get_fname(i,type,row,col,fname)
int i;
int type;
int row;
int col;
char *fname;
{
  char text1[39],text2[39];  /* error message text              */
  int j, k=0;                /* counters                        */
  char ch;

  cursor_on();               /* turn on cursor                  */
  goto_xy(row,col);          /* move cursor to start of box     */

  /* get user input */
  screen_getstrg(i,row,col,fname,NORM_VID,input);
  strstrip(fname);

  /* check for quit option */
  if (fname[0]=='Q')
  {
    cursor_off();
    goto_xy(0,0);
    return ;
  }

  /* check for valid filename */

  while ( (ch=fname[k]) != NULL )
  {
    if ( (ch>='a' && ch<='z') || (ch>='A' && ch<='Z')
          || (ch>='0' && ch<='9') || ch == '_' || ch == '-')
    {
      k++;
      continue;
    }
    else
    {
      sprintf(text1,"File name contains invalid characters");
      sprintf(text2,"==> please try again <==");
      display_error(ERROR,text1,text2);
      vid_box(row,col,9);
      get_fname(i,type,row,col,fname);
      return ;
    }
  }
```

```
  if (type==0)          /* tug */
  {
    sprintf(text1,"%s.tug",fname);
    slctouc(text1,text2);
    strncpy(fname,text2,12);
    strstrip(fname);
  }
  else                  /* tow */
  {
    sprintf(text1,"%s.tow",fname);
    slctouc(text1,text2);
    strncpy(fname,text2,12);
    strstrip(fname);
  }

  cursor_off();
  goto_xy(0,0);
}

/*****************************************************************
  This function writes the tug data to a user specified file.
*****************************************************************/
void save_tug_file(startrow,startcol,endrow,endcol,tug,tug_data)
int startrow,startcol,endrow,endcol;
char *tug;
float *tug_data;
{
  FILE *out;
  char fname[13];
  char string[25];
  char text1[39];
  int start;

  /* prompt for file name */
  start = startcol + 3;
  write_string(endrow-3,start,footer[6],REV_VID);
  vid_box(endrow-3,start+strlen(footer[6])+3,9);

  get_fname(6,0,endrow-3,start+strlen(footer[6])+3,fname);
  out=fopen(fname,"w");

  if (out==NULL)
  {
    sprintf(text1,"Can't open file %s",fname);
    display_error(ERROR,text1," ");
    save_tug_file(startrow,startcol,endrow,endcol,tug,tug_data);
    return;
  }
  else
  {
    /* write data to file */
    fprintf(out,"! Tug data file\n");
    fprintf(out,"! Tug class:\n");
    fprintf(out,"%s\n",tug);
```

```
    fprintf(out,"! Hawser diameter (inches):\n");
    sprintf(string,"%-.2f\n",tug_data[1]);
    fprintf(out,"%s",string);

    fprintf(out,"! Hawser scope (feet):\n");
    sprintf(string,"%-.1f\n",tug_data[2]);
    fprintf(out,"%s",string);

    fprintf(out,"! Chain pendant diameter (inches):\n");
    sprintf(string,"%-.2f\n",tug_data[3]);
    fprintf(out,"%s",string);

    fprintf(out,"! Chain scope (feet):\n");
    sprintf(string,"%-.2f\n",tug_data[4]);
    fprintf(out,"%s",string);
  }
  fclose(out);
}
```

```
/*****************************************************************************
  File: tow.c
  Author: Todd J. Peltzer
  Last update: 17 April 1989

       This file contains functions which support the option "Select Tow"
  in the main program.
-----------------------------------------------------------------------------
  Functions:
    get_tow_data()
    get_data()
    get_ship_file()
    save_ship_file()

*****************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "keydef.h"
#include "video.h"

static char *label[] =
{
  "TOW DATA",
  "Hull no:",
  "Full load displacement:",
  "Tow speed:",
  "Max expected wind spd:",
  "Rel wind direction:",
  "Propeller status:"
};

static char *label1[] =
{
  "Options",
  "Retrieve File",
  "Enter name of tow file",
  "to retrieve (8 char max): "
};

static char *units[] =
{
  "tons",
  "kts",
  "kts",
  "deg"
};

static char *footer[] =
{
  "Please enter data.",
  "Is all data correct? (yes/no): ",
  "F1 Hull F2 Disp F3 Tow F4 Wind",
  "F5 Rel wind F6 Prop status",
  " Press INS to continue ",
```

```
  "  Save data to file? (yes/no): ",
  " Enter tow file name: "
};

char *menu[]= {
   "Locked  ",
   "Trailing",
   "Removed "
};

static char choice[4];

typedef struct
{
  int startcol;
  int endcol;
} data_box;

static data_box input[] =
{
  {32,55},        /* hull no        */
  {47,54},        /* displacement   */
  {47,54},        /* tow speed      */
  {47,54},        /* wind speed     */
  {47,54},        /* wind direction */
  {54,58},        /* yes/no choice  */
  {49,58}         /* tow file name  */
};

void get_data(),save_ship_file();

/*****************************************************************************
     This function prompts the user for data concerning the tow. Input items
  are: hull number, displacement, tow speed, wind speed, relative wind
  direction, and propeller status. A popup menu is used to select propeller
  status.
*****************************************************************************/
get_tow_data(flag,hull_no,class,tow_data,ship_data)
int *flag;
char *hull_no;
char *class;
float *tow_data;
float *ship_data;
{
  int startrow=2;
  int startcol=20;
  int endrow=22;
  int endcol=60;

  int row, col, i, start, start1, key;
  int status,choice1;
  char string[25],text1[39],text2[39];

  if (!flag[1])         /* tow data does not exist */
```

```
{
  /* initialize data variables to zero */
  hull_no[0] = NULL;
  for (i=0; i<5; i++)
    tow_data[i] = 0.0;
}

/* clear background */
clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

/* write header */
write_header(startrow,startcol,endcol,label[0],REV_VID);

if (!flag[1] && !flag[2])       /* enter new data */
{
  clear(startrow+5,startcol+1,endrow-3,endcol-1,REV_VID);
}
else if (flag[1] && !flag[2]) ;  /* edit existing data */
else if (flag[2])    /* retrieve data from file */
{
  status=get_ship_file(startrow,startcol,endrow,endcol,hull_no,class,
              tow_data,ship_data,flag);
  if (status<0)
  {
    cursor_off();
    flag[2]=0;
    return (-1);
  }
}
clear(startrow+5,startcol+1,endrow-3,endcol-1,REV_VID);

/* write data labels */
for(row=startrow+5,i=1; i<=6; i++,row += 2)
  write_string(row,startcol+3,label[i],REV_VID);

/* write units */
for(row=startrow+7,i=0; i<4; i++,row += 2)
  write_string(row,endcol-5,units[i],REV_VID);

if (flag[1] || flag[2])
{
  /* display data */
  /* hull number */
  write_string(startrow+5,startcol+2+strlen(label[1])+2,hull_no,REV_VID);

  /* displacement */
  sprintf(string,"%6.0f",tow_data[0]);
  write_string(startrow+7,startcol+2+strlen(label[2])+2,string,REV_VID);

  /* tow speed */
  sprintf(string,"%6.1f",tow_data[1]);
  write_string(startrow+9,startcol+2+strlen(label[3])+15,string,REV_VID);

  /* wind speed */
```

```
    sprintf(string,"%6.1f",tow_data[2]);
    write_string(startrow+11,startcol+2+strlen(label[4])+3,string,REV_VID);

    /* relative wind direction */
    sprintf(string,"%6.1f",tow_data[3]);
    write_string(startrow+13,startcol+2+strlen(label[5])+6,string,REV_VID);

    /* prop status */
    if (tow_data[4] == LOCKED)
    {
      write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[0],REV_VID);
    }
    else if (tow_data[4] == TRAILING)
    {
      write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[1],REV_VID);
    }
    else if (tow_data[4] == REMOVED)
    {
      write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[2],REV_VID);
    }
  }
  else if (!flag[1])
  {
    /* write footer */
    start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
    write_string(startrow+17,start,footer[0],REV_VID);

    /* create normal video boxes for data entry */
    vid_box(startrow+5,startcol+2+strlen(label[1])+2,23);
    vid_box(startrow+7,startcol+2+strlen(label[2])+2,7);
    vid_box(startrow+9,startcol+2+strlen(label[3])+15,7);
    vid_box(startrow+11,startcol+2+strlen(label[4])+3,7);
    vid_box(startrow+13,startcol+2+strlen(label[5])+6,7);

    /*****************************************************************/
    /*                      get user input                           */
    /*****************************************************************/
    /* get hull_no */
    get_data(0,startrow+5,startcol+2+strlen(label[1])+2,hull_no,tow_data);
    for (i=0; i<23; i++)
      write_char(startrow+5,startcol+2+strlen(label[1])+2+i,' ',REV_VID);
    write_string(startrow+5,startcol+2+strlen(label[1])+2,hull_no,REV_VID);

    /* get displacement */
    get_data(1,startrow+7,startcol+2+strlen(label[2])+2,hull_no,tow_data);
    for (i=0; i<7; i++)
      write_char(startrow+7,startcol+2+strlen(label[2])+2+i,' ',REV_VID);
    sprintf(string,"%6.0f",tow_data[0]);
    write_string(startrow+7,startcol+2+strlen(label[2])+2,string,REV_VID);

    /* get tow speed */
    get_data(2,startrow+9,startcol+2+strlen(label[3])+15,hull_no,tow_data);
    for (i=0; i<7; i++)
      write_char(startrow+9,startcol+2+strlen(label[3])+15+i,' ',REV_VID);
```

```
  sprintf(string,"%6.1f",tow_data[1]);
  write_string(startrow+9,startcol+2+strlen(label[3])+15,string,REV_VID);

  /* get wind speed */
  get_data(3,startrow+11,startcol+2+strlen(label[4])+3,hull_no,tow_data);
  for (i=0; i<7; i++)
    write_char(startrow+11,startcol+2+strlen(label[4])+3+i,' ',REV_VID);
  sprintf(string,"%6.1f",tow_data[2]);
  write_string(startrow+11,startcol+2+strlen(label[4])+3,string,REV_VID);

  /* get relative wind direction */
  get_data(4,startrow+13,startcol+2+strlen(label[5])+6,hull_no,tow_data);
  for (i=0; i<7; i++)
    write_char(startrow+13,startcol+2+strlen(label[5])+6+i,' ',REV_VID);
  sprintf(string,"%6.1f",tow_data[3]);
  write_string(startrow+13,startcol+2+strlen(label[5])+6,string,REV_VID);

  /* get propeller status */
  tow_data[4] = popup(menu,"LTR",3,startrow+13,endcol-13,SINGLE,REV_VID,0);
  if (tow_data[4] == LOCKED)
  {
    write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[0],REV_VID);
  }
  else if (tow_data[4] == TRAILING)
  {
    write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[1],REV_VID);
  }
  else if (tow_data[4] == REMOVED)
  {
    write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[2],REV_VID);
  }

  /* erase footer */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  for (i=0; i<strlen(footer[0]); i++)
    write_char(startrow+17,start+i,' ',REV_VID);
}

/*****************************************************************/
/*    prompt user for confirmation of data; give edit option     */
/*****************************************************************/
start = startcol + 3;
write_string(startrow+17,start,footer[1],REV_VID);
vid_box(startrow+17,start+strlen(footer[1]),4);
get_data(5,startrow+17,start+strlen(footer[1]),hull_no,tow_data);

/* test if data correct */
if ( choice[0]!='y' && choice[0]!='Y' && choice[0]!=NULL )
{
  /* erase previous message */
  start = startcol + 3;
  for (i=0; i<strlen(footer[1])+4; i++)
    write_char(endrow-3,start+i,' ',REV_VID);
```

```
/* write quit message */
start=(endcol-startcol-strlen(footer[4]))/2 + startcol;
write_string(endrow,start,footer[4],REV_VID);

/* write first line of edit "menu" */
start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
write_string(endrow-3,start,footer[2],REV_VID);

/* write function key indicators in NORM_VID */
for(i=0; i<2; i++)
{
  write_char(endrow-3,start+i,footer[2][i],NORM_VID);
  write_char(endrow-3,start+8+i,footer[2][i+8],NORM_VID);
  write_char(endrow-3,start+16+i,footer[2][i+16],NORM_VID);
  write_char(endrow-3,start+23+i,footer[2][i+23],NORM_VID);
}

/* write second line of edit "menu" */
start1=(endcol-startcol-strlen(footer[3]))/2 + startcol;
write_string(endrow-2,start1,footer[3],REV_VID);

/* write function key indicators in NORM_VID */
for(i=0; i<2; i++)
{
  write_char(endrow-2,start1+i,footer[3][i],NORM_VID);
  write_char(endrow-2,start1+12+i,footer[3][i+12],NORM_VID);
}

/*****************************************************************/
/*                          edit data                            */
/*****************************************************************/
/* move cursor to F1 highlight */
cursor_on();
goto_xy(startrow+17,start);

/* get user's choice */
while (1)
{
  key=get_special();
                  /***************/
  if (key==F1)    /* hull number */
  {               /***************/
    vid_box(startrow+5,startcol+2+strlen(label[1])+2,23);
    get_data(0,startrow+5,startcol+2+strlen(label[1])+2,hull_no,tow_data);
    for (i=0; i<23; i++)
      write_char(startrow+5,startcol+2+strlen(label[1])+2+i,' ',REV_VID);
    write_string(startrow+5,startcol+2+strlen(label[1])+2,hull_no,REV_VID);
    cursor_on();
    goto_xy(endrow-3,start);
  }                 /****************/
  else if (key==F2) /* displacement */
  {                 /****************/
    vid_box(startrow+7,startcol+2+strlen(label[2])+2,7);
    get_data(1,startrow+7,startcol+2+strlen(label[2])+2,hull_no,tow_data);
```

```
  for (i=0; i<7; i++)
    write_char(startrow+7,startcol+2+strlen(label[2])+2+i,' ',REV_VID);
  sprintf(string,"%6.0f",tow_data[0]);
  write_string(startrow+7,startcol+2+strlen(label[2])+2,string,REV_VID);
  cursor_on();
  goto_xy(endrow-3,start+8);
}                    /***************/
else if (key==F3)    /* tow speed   */
{                    /***************/
  vid_box(startrow+9,startcol+2+strlen(label[3])+15,7);
  get_data(2,startrow+9,startcol+2+strlen(label[3])+15,hull_no,tow_data);
  for (i=0; i<7; i++)
    write_char(startrow+9,startcol+2+strlen(label[3])+15+i,' ',REV_VID);
  sprintf(string,"%6.1f",tow_data[1]);
  write_string(startrow+9,startcol+2+strlen(label[3])+15,string,REV_VID);
  cursor_on();
  goto_xy(endrow-3,start+16);
}                    /***************/
else if (key==F4)    /* wind speed  */
{                    /***************/
  vid_box(startrow+11,startcol+2+strlen(label[4])+3,7);
  get_data(3,startrow+11,startcol+2+strlen(label[4])+3,hull_no,tow_data);
  for (i=0; i<7; i++)
    write_char(startrow+11,startcol+2+strlen(label[4])+3+i,' ',REV_VID);
  sprintf(string,"%6.1f",tow_data[2]);
  write_string(startrow+11,startcol+2+strlen(label[4])+3,string,REV_VID);
  cursor_on();
  goto_xy(endrow-3,start+23);
}                    /****************/
else if (key==F5)    /* rel wind dir */
{                    /****************/
  vid_box(startrow+13,startcol+2+strlen(label[5])+6,7);
  get_data(4,startrow+13,startcol+2+strlen(label[5])+6,hull_no,tow_data);
  for (i=0; i<7; i++)
    write_char(startrow+13,startcol+2+strlen(label[5])+6+i,' ',REV_VID);
  sprintf(string,"%6.1f",tow_data[3]);
  write_string(startrow+13,startcol+2+strlen(label[5])+6,string,REV_VID);
  cursor_on();
  goto_xy(endrow-2,start1);
}                    /***************/
else if (key==F6)    /* prop status */
{                    /***************/
  cursor_off();
  tow_data[4] = popup(menu,"LTR",3,startrow+13,endcol-13,SINGLE,REV_VID,0);
  if (tow_data[4] == LOCKED)
  {
    for (i=0; i<=strlen(menu[2]); i++)
      write_char(startrow+15,startcol+2+strlen(label[6])+10+i,' ',REV_VID);
    write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[0],REV_VID);
  }
    else if (tow_data[4] == TRAILING)
  {
    for (i=0; i<=strlen(menu[2]); i++)
      write_char(startrow+15,startcol+2+strlen(label[6])+10+i,' ',REV_VID);
```

```
        write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[1],REV_VID);
      }
        else if (tow_data[4] == REMOVED)
      {
        for (i=0; i<=strlen(menu[2]); i++)
          write_char(startrow+15,startcol+2+strlen(label[6])+10+i,' ',REV_VID);
        write_string(startrow+15,startcol+2+strlen(label[6])+10,menu[2],REV_VID);
      }
      cursor_on();
      goto_xy(endrow-2,start1+12);
    }                          /***************/
    else if (key==INSERT)      /* quit edit   */
    {                          /***************/
      /* erase menu */
      for (i=0; i<strlen(footer[2]); i++)
        write_char(startrow+17,start+i,' ',REV_VID);
      for (i=0; i<strlen(footer[3]); i++)
        write_char(startrow+18,start1+i,' ',REV_VID);

      /* erase message */
      start = (endcol-startcol-strlen(footer[4]))/2 + startcol;
      for (i=0; i<strlen(footer[4]); i++)
        write_char(endrow,start+i,205,REV_VID);
      break;
    }
  )
  cursor_off();
  )              /************************/
  else           /* editing not required */
  {              /************************/
    /* erase previous message */
    start = startcol + 3;
    for (i=0; i<strlen(footer[1])+4; i++)
      write_char(endrow-3,start+i,' ',REV_VID);
  )
  return (0);
)

/*****************************************************************************
     This function gets the user's input for the data requested in the
  function get_tow_data(). Uses screen_getstring() for screen I/O. Some data
  checking is performed for each data type.
*****************************************************************************/
void get_data(i,row,col,hull_no,data)
int i;
int row;
int col;
char *hull_no;
float *data;
{
  int nbr;                    /* required arg for stofa       */
  int status;                 /* takes return value for stofa */
  char string[24];            /* input string                 */
  char text1[39], text2[39]; /* error message text           */
```

```
int j, k;                    /* counters                        */
int length;

cursor_on();                 /* turn on cursor                  */
goto_xy(row,col);            /* move cursor to start of box     */

/* get user input */
screen_getstrg(i,row,col,string,NORM_VID,input);

if (i==0)
{
  /* convert input string to upper case; output to hull_no */
  slctouc(string,hull_no);

  /* ensure hull number is in proper format */
  /* -- first copy hull_no */
  strcpy(string,hull_no);

  if (hull_no[0]=='T')
  {
    if (hull_no[1]==' ')
    {
      hull_no[1] = '-';
      strcpy(string,hull_no);
    }
    else if (hull_no[1] != '-')
    {
      string[1] = '-';
      length = strlen(hull_no);
      for (k=2; k<=length; k++)
        string[k] = hull_no[k-1];
      string[k] = NULL;
      strcpy(hull_no,string);
    }
  }
  length=strlen(hull_no);
  for (j=0; j<length; j++)
  {
    if (hull_no[j]>='0' && hull_no[j]<='9')
    {
      if (hull_no[j-1] == '-')
      {
        hull_no[j-1] = ' ';
        break;
      }
      else if (hull_no[j-1] != ' ')
      {
        string[j] = ' ';
        for (k=j+1; k<=strlen(hull_no); k++)
          string[k] = hull_no[k-1];
        string[k] = NULL;
        slctouc(string,hull_no);
        break;
      }
```

```
      else
      {
        break;
      }
    }
  }

}
else if (i>0 && i<4)
{
  /* check for valid numeric input */
  for (k=0; k<strlen(string); k++)
  {
    if ( string[k]=='.' ) ;
    else if (string[k]<'0' || string[k]>'9')
    {
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
      vid_box(row,col,7);
      get_data(i,row,col,hull_no,data);
      return ;
    }
  }
  /* if input valid numbers only, convert string to float */
  status = stofa(string,&data[i-1],&nbr,1);
}
else if (i==4)
{
  /* check for valid numeric input */
  for (k=0; k<strlen(string); k++)
  {
    if ( string[k]=='.' ) ;
    else if (string[k]=='-') ;
    else if (string[k]<'0' || string[k]>'9')
    {
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
      vid_box(row,col,7);
      get_data(i,row,col,hull_no,data);
      return ;
    }
  }
  /* if input valid numbers only, convert string to float */
  status = stofa(string,&data[i-1],&nbr,1);
}
else      /* i == 5 */
{
  strcpy(choice,string);
}

/*********************************************************/
/*            check data for consistency                 */
/*********************************************************/
               /***************/
```

```
if (i==0)       /* hull number */
{               /****************/
  if (hull_no[0]<'A' || 'Z'<hull_no[0] && hull_no[0]<'a' || hull_no[0]>'z')
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,23);
    get_data(i,row,col,hull_no,data);
    return ;
  }
}               /****************/
if (i==1}       /* displacement */
{               /****************/
  if (data[i-1]<=0.0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_data(i,row,col,hull_no,data);
    return ;
  }
  else if ( data[i-1]<350.0 || data[i-1]>91000.0 && data[i-1]<=100000.0 )
  {
    sprintf(text1,"Displacement is outside range");
    sprintf(text2,"of data in Table G-2");
    display_error(WARN,text1,text2);
  }
  else if ( data[i-1]>100000.0 )
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7};
    get_data(i,row,col,hull_no,data);
    return ;
  }
}               /****************/
if (i==2)       /* towing speed */
{               /****************/
  if (data[i-1]<=0.0}
  {
    sprintf(text1,"Invalid entry; try again"};
    display_error(ERROR,text1," ");
    vid_box(row,col,7};
    get_data(i,row,col,hull_no,data);
    return ;
  }
  else if (data[i-1]>10.0 && data[i-1]<=12.0)
  {
    sprintf(text1,"Planned tow speed is higher than"};
    sprintf(text2,"normally recommended");
    display_error(WARN,text1,text2);
  }
  else if (data[i-1]>12.0}
  {
```

```
    sprintf(text1,"Outside range of Fig. G-1");
    sprintf(text2,"=>please enter new tow speed<=");
    display_error(ERROR,text1,text2);
    vid_box(row,col,7);
    get_data(i,row,col,hull_no,data);
    return ;
  }
}             /***************/
if (i==3)     /* wind speed */
{             /***************/
  if (data[i-1]<0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_data(i,row,col,hull_no,data);
    return ;
  }
  else if (data[i-1]>48.0)
  {
    sprintf(text1,"Outside range of Fig. G-2");
    sprintf(text2,"=>please enter new wind speed<=");
    display_error(ERROR,text1,text2);
    vid_box(row,col,7);
    get_data(i,row,col,hull_no,data);
    return ;
  }
}             /*******************/
if (i==4)     /* wind direction */
{             /*******************/
  if (data[i-1]<0)
  {
    sprintf(text1,"Use only positive angles");
    display_error(WARN,text1," ");
    data[i-1] = -data[i-1];
  }
  else if (data[i-1]>180.0 && data[i-1]<360.0)
  {
    sprintf(text1,"Use angles between 0 and 180");
    display_error(WARN,text1," ");
    data[i-1] = 360.0 - data[i-1];
  }
  else if (data[i-1]>70.0 && data[i-1]<110.0)
  {
    sprintf(text1,"No extreme tension data for beam seas");
    sprintf(text2,"==> use angles < 70 or > 110 <==");
    display_error(ERROR,text1,text2);
    vid_box(row,col,7);
    get_data(i,row,col,hull_no,data);
    return ;
  }
  else if (data[i-1]>360.0)
  {
    sprintf(text1,"Invalid entry; try again");
```

```
      display_error(ERROR,text1," ");
      vid_box(row,col,7);
      get_data(i,row,col,hull_no,data);
      return ;
    )
  }

  cursor_off();
  goto_xy(0,0);
)

/*****************************************************************
  This function retrieves tow data from a user specified file.
*****************************************************************/
get_ship_file(startrow,startcol,endrow,endcol,hull_no,class,
              tow_data,ship_data,flag)
int startrow,startcol;
int endrow,endcol;
char *hull_no;
char *class;
float *tow_data;
float *ship_data;
int *flag;
{
  FILE *in;
  char fname[13];
  char inline[81];
  char text1[39],text2[39];
  int i,j,status,nbr;
  float array[2];
  int start;

  start = (endcol-startcol-strlen(label1[1]))/2 + startcol;
  write_string(startrow+8,start,label1[1],REV_VID);
  for (i=0; i<2; i++)      /* write prompt */
  {
    start = startcol+3;
    write_string(startrow+10+i,start,label1[i+2],REV_VID);
  )
  vid_box(startrow+11,start+strlen(label1[3]),9);
  sprintf(text1,"Type 'Q' to quit");
  start = (endcol-startcol-strlen(text1))/2 + startcol;
  write_string(endrow-3,start,text1,REV_VID);
  get_fname(6,1,startrow+11,startcol+3+strlen(label1[3]),fname);

  /* check for quit option */
  if (fname[0]=='Q')
  {
    cursor_off();
    goto_xy(0,0);
    return (-1);
  )

  in=fopen(fname,"r");
```

```
if (in==NULL)
{
  sprintf(text1,"Can't open file %s",fname);
  display_error(ERROR,text1," ");
  clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
  get_ship_file(startrow,startcol,endrow,endcol,hull_no,
              class,tow_data,ship_data,flag);
  return (0);
}
else
{
  /* read data from file */
  fgets(inline,81,in);
  strstrip(inline);

  /* test for valid tow file */
  if (!(strncmp(inline,"! Tow data file",15)))
  {
    fgets(inline,81,in);
    fgets(inline,81,in);
    strstrip(inline);

    /* test if ship file */
    if (strncmp(inline,"SHIP",4))
    {
      sprintf(text1,"%s is not a ship data file!",fname);
      display_error(ERROR,text1," ");
      clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
      get_ship_file(startrow,startcol,endrow,endcol,hull_no,
              class,tow_data,ship_data,flag);
      return (0);
    }

    /* get hull number */
    fgets(inline,81,in);
    fgets(inline,81,in);
    strstrip(inline);
    strncpy(hull_no,inline,24);
    strstrip(hull_no);

    /* get class */
    fgets(inline,81,in);
    fgets(inline,81,in);
    strstrip(inline);
    strncpy(class,inline,24);
    strstrip(class);

    /* read remaining tow data */
    for (i=0; i<5; i++)
    {
      fgets(inline,81,in);
      fgets(inline,81,in);
      status=stofa(inline,array,&nbr,1);
      tow_data[i]=array[0];
```

```
      }

      /* read ship data */
      for (i=0; i<6; i++)
      {
        fgets(inline,81,in);
        fgets(inline,81,in);
        status=stofa(inline,array,&nbr,1);
        ship_data[i]=array[0];
      }
      fclose(in);
    }
    else
    {
      sprintf(text1,"%s is not a tow data file!",fname);
      display_error(ERROR,text1," ");
      clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
      get_ship_file(startrow,startcol,endrow,endcol,hull_no,
                class,tow_data,ship_data,flag);
      return (0);
    }
  }
}

/*****************************************************************
  This function writes the tow data for self-propelled ships to
  a user specified file.
*****************************************************************/
void save_ship_file(startrow,startcol,endrow,endcol,hull_no,
                class,tow_data,ship_data)
int startrow,startcol,endrow,endcol;
char *hull_no;
char *class;
float *tow_data;
float *ship_data;
{
  FILE *out;
  char fname[13];
  char string[25];
  char text1[39];
  int start;

  /* prompt for file name */
  start = startcol + 3;
  write_string(endrow-2,start,footer[6],REV_VID);
  vid_box(endrow-2,start+strlen(footer[6])+3,9);

  get_fname(6,1,endrow-2,start+strlen(footer[6])+3,fname);
  out=fopen(fname,"w");

  if (out==NULL)
  {
    sprintf(text1,"Can't open file %s",fname);
    display_error(ERROR,text1," ");
```

```
    save_ship_file(startrow,startcol,endrow,endcol,hull_no,class,
                   tow_data,ship_data);
    return;
  }
  else
  {
    /* write tow data to file */
    fprintf(out,"! Tow data file\n");
    fprintf(out,"! Tow type:\n");
    fprintf(out,"SHIP\n");

    fprintf(out,"! Hull number:\n");
    fprintf(out,"%s\n",hull_no);

    fprintf(out,"! Ship class:\n");
    fprintf(out,"%s\n",class);

    fprintf(out,"! Ship displacement:\n");
    sprintf(string,"%-.1f\n",tow_data[0]);
    fprintf(out,"%s",string);

    fprintf(out,"! Tow speed (kts):\n");
    sprintf(string,"%-.1f\n",tow_data[1]);
    fprintf(out,"%s",string);

    fprintf(out,"! Wind speed (kts):\n");
    sprintf(string,"%-.2f\n",tow_data[2]);
    fprintf(out,"%s",string);

    fprintf(out,"! Relative wind dir:\n");
    sprintf(string,"%-.2f\n",tow_data[3]);
    fprintf(out,"%s",string);

    fprintf(out,"! Propeller status:\n");
    sprintf(string,"%-.0f\n",tow_data[4]);
    fprintf(out,"%s",string);

    /* write ship data to file */
    fprintf(out,"! Tabulated displacement:\n");
    sprintf(string,"%-.0f\n",ship_data[0]);
    fprintf(out,"%s",string);

    fprintf(out,"! Frontal windage area:\n");
    sprintf(string,"%-.0f\n",ship_data[1]);
    fprintf(out,"%s",string);

    fprintf(out,"! Wind coefficient:\n");
    sprintf(string,"%-.2f\n",ship_data[2]);
    fprintf(out,"%s",string);

    fprintf(out,"! Propeller area:\n");
    sprintf(string,"%-.2f\n",ship_data[3]);
    fprintf(out,"%s",string);
```

```
    fprintf(out,"! Hull resistance curve number:\n");
    sprintf(string,"%-.0f\n",ship_data[4]);
    fprintf(out,"%s",string);

    fprintf(out,"! Wave resistance curve number:\n");
    sprintf(string,"%-.0f\n",ship_data[5]);
    fprintf(out,"%s",string);
  }
  fclose(out);
}
```

```
/*****************************************************************************
  File: tab.c
  Author: Todd J. Peltzer
  Last Update : 30 April 1989

    This file contains functions which support the display of Table G-2
  of the USN Towing Manual. The user can scroll through the choices
  in the table to select a ship. Once a selection is made, the user is
  prompted to confirm the choice.
-----------------------------------------------------------------------------
  Functions:
    display_tab_g2()
    up_line()
    down_line()
    up_page()
    down_page()
    goto_home()
    goto_end()
    read_table()
    read_data()
    match_ship()
    match_type()
    match_nbr()
    read_class()
*****************************************************************************/

#include "stdio.h"
#include "dos.h"
#include "stdlib.h"
#include "keydef.h"   /* Define aux byte values for IBM keyboard */
#include "video.h"

void up_line(),down_line(),up_page(),down_page(),goto_home(),goto_end();
void read_table(), read_data(), read_class();
char far *vid_mem;

static char *page_header[2] =
{
"                                                              WINDAGE   PROP",
"         CLASS                    DESCRIPTION              DISP    AREA     AREA"
};

extern char *hull_no;
extern char *class;

static char *footer[2] =
{
  "(Items marked with 'e' are best estimate.)",
  "Use arrow keys, [pgup], [pgdn], [home], [end] to view choices."
);

union inkey
(
  char ch[2];
```

```
  int i;
) c;

/*****************************************************************************
     Display the table . . .
*****************************************************************************/
display_tab_g2(table,data,hull_no,class)
char **table;
float *data;
char *hull_no;
char *class;
{
  FILE *in;
  int i, j, len, start, status;
  int arrow_choice=0;
  int n=141;
  int line_no=0;
  int match, type;
  int num;
  char string[81], ch;

  set_video();
  cls();
  cursor_off();    /* turn off blinking cursor */

  /* print header lines at top of screen */
  for (i=0; i<2; i++)
    write_string(i,0,page_header[i],NORM_VID);

  /* draw border */
  for (j=0; j<80; j++)
    write_char(2,j,205,NORM_VID);

  /* write footer */
  for (i=0; i<2; i++)
  {
    len = strlen(footer[i]);
    start = (80 - len)/2;
    write_string(23+i,start,footer[i],NORM_VID);
  }

  /*******************************************************/
  /* find match with hull number of ship entered by user */
  /*******************************************************/
  match = match_ship(hull_no,table,&type);

  if (match<0 && type>=0)   /* hull type matches; hull no. does not */
    match = type;

  if (match>=0)
  {
    /* set current choice to "match" */
    arrow_choice=match;
    num=n-1-arrow_choice;
```

```
    if (arrow_choice==n-1)
      goto_end(&arrow_choice,&line_no,n,table);

    else if (num < 20)
    {
      /* write first screen */
      for (i=0; i<=num; i++)
        write_string(i+3, 0, table[n-1-num+i], NORM_VID);

      /* highlight match */
      write_string(3, 0, table[arrow_choice], REV_VID);

      /* ensure rest of screen is blank */
      clear (num+1+3,0,22,79);
    }

    else
    {
      /* write first screen */
      for (i=0; i<20; i++)
        write_string(i+3,0,table[arrow_choice+i],NORM_VID);

      /* highlight first entry */
      write_string(3,0,table[arrow_choice],REV_VID);
    }
  }
  else
  {
    /* write first screen */
    for (i=0; i<20; i++)
      write_string(i+3,0,table[i],NORM_VID);

    /* highlight first entry */
    write_string(3,0,table[0],REV_VID);
  }

  /*********************************************************/
  /*            get user's response                        */
  /*********************************************************/
  status = get_tab_g2_resp(&arrow_choice,&line_no,n,table);

  if(!status)  /* status==0 is normal return from get_tab_g2_resp */
  {
    /* read data from appropriate file */
    in = fopen("table.dat","r");
    read_data(in,arrow_choice,data,6);
    read_class(arrow_choice,table,class,23);
  }
  /* need else statement to handle situation if escape key is pressed */

}

/*****************************************************************************
```

```
    This function allows the user to scroll through choices and make
  selections. Returns 0 if selection is made; returns -1 if escape key
  is pressed.
*****************************************************************************/
get_tab_g2_resp(arrow_choice,line_no,n,table)
int *arrow_choice;          /* current table entry        */
int *line_no;               /* current screen line number */
int  n;                     /* number of entries in table */
char **table;               /* current ship table         */
{
  for (;;)
  {
    while(!bioskey(1)) ; /* wait for key stroke */
    c.i = bioskey(0);    /* read the key */

    if(c.ch[0])          /* is normal key */
    {
      switch(c.ch[0])
      {
        case '\r':
          return 0;
        case ESC :
          return -1;     /* cancel */
        default :
          beep1();
          break;
      }                  /* end "switch" */
    }                    /* end "if" */
    else
    {  /* is special key */
      switch(c.ch[1])
      {
        case UP_ARROW :
          up_line(arrow_choice,line_no,n,table);
          break;
        case DOWN_ARROW :
          down_line(arrow_choice,line_no,n,table);
          break;
        case PAGE_UP :
          up_page(arrow_choice,line_no,n,table);
          break;
        case PAGE_DOWN :
          down_page(arrow_choice,line_no,n,table);
          break;
        case HOME :
          goto_home(arrow_choice,line_no,n,table);
          break;
        case END :
          goto_end(arrow_choice,line_no,n,table);
          break;
        default :
          beep1();
          break;
      }                  /* end "switch" */
```

```
    )                       /* end "else" */
  )
}

/*****************************************************************************
   This function moves the highlighted choice up one line.
     -- if beginning of table, does nothing;
     -- if top of screen but not beginning of table, scrolls screen;
     -- otherwise, just moves up one line.
*****************************************************************************/
void up_line(arrow_choice,line_no,n,table)
int *arrow_choice;            /* current table entry         */
int *line_no;                 /* current screen line number  */
int  n;                       /* number of entries in table  */
char **table;             /* current ship table          */
{
  int i,num;

  if (*arrow_choice == 0) ;          /* first table entry */

  else if (*line_no == 0)            /* top of window, not first entry */
  {
    /* highlight previous entry */
    write_string(3, 0, table[*arrow_choice-1], REV_VID);

    /* is this the last screen? */
    num=n-*arrow_choice;
    if (num<20)
    {
      for (i=0; i<num; i++)
        write_string(i+4, 0, table[*arrow_choice+i], NORM_VID);
    }
    else
    {
      for (i=0; i<19; i++)
        write_string(i+4, 0, table[*arrow_choice+i], NORM_VID);
    }
      (*arrow_choice)--;    /* line_no stays the same */
  }

  else
  {
    /* highlight previous entry */
    write_string(*line_no-1+3, 0, table[*arrow_choice-1], REV_VID);

    /* restore last highlighted entry to normal video */
    write_string(*line_no+3, 0, table[*arrow_choice], NORM_VID);

    /* decrement table entry # and screen line # */
    (*arrow_choice)--;
    (*line_no)--;
  }
}
```

```
/*****************************************************************************
   This function moves the highlighted choice down one line.
     -- if end of table, does nothing;
     -- if bottom of screen but not end of table, scrolls screen down one line;
     -- otherwise, just moves down one line.
*****************************************************************************/
void down_line(arrow_choice,line_no,n,table)
int *arrow_choice;          /* current table entry        */
int *line_no;               /* current screen line number */
int  n;                     /* number of entries in table */
char **table;               /* ship table                 */
{
  int i;

  if (*arrow_choice == n-1) ;        /* last table entry; do nothing */

  else if (*line_no == 19)           /* end of window, not last entry */
  {
    /* display table */
    for (i=0; i<19; i++)
      write_string(i+3, 0, table[*arrow_choice-18+i], NORM_VID);

    /* highlight next entry */
    write_string(22, 0, table[*arrow_choice+1], REV_VID);

    (*arrow_choice)++;               /* line_no stays the same */
  }

  else                               /* move highlight down one line */
  {
    /* highlight next entry */
    write_string(*line_no+1+3, 0, table[*arrow_choice+1], REV_VID);

    /* restore last highlighted entry to normal video */
    write_string(*line_no+3, 0, table[*arrow_choice], NORM_VID);

    (*arrow_choice)++;
    (*line_no)++;
  }
)

/*****************************************************************************
   This function displays previous page of table.
     -- if beginning of table, does nothing;
     -- if first page, moves highlight to first line;
     -- otherwise, highlight stays on same screen line.
*****************************************************************************/
void up_page(arrow_choice,line_no,n,table)
int *arrow_choice;          /* current table entry        */
int *line_no;               /* current screen line number */
int  n;                     /* number of entries in table */
char **table;               /* ship table                 */
{
  int i, last;
```

```
  if (*arrow_choice == 0) ;         /* first table entry */

  /* if on first page, but not first entry, highlight first entry */
  else if (*arrow_choice < 20)
  {
    for (i=0; i<20; i++)
      write_string(i+3, 0, table[i], NORM_VID);
    /* highlight first entry */
    write_string(3, 0, table[0], REV_VID);

    *arrow_choice = 0;     /* reset */
    *line_no = 0;
  }

  /* not on first page; display previous page */
  else
  {
    /* compute index of top of previous page */
    last = *arrow_choice - *line_no - 20;

    /* display previous page */
    if (last<=0)     /* jumping 20 entries would go to or past top of table */
      {
        for (i=0; i<20; i++)
          write_string(i+3, 0, table[i], NORM_VID);
        write_string(*line_no+last+3,0,table[*arrow_choice-20],REV_VID);
        *line_no=0;          /* reset line_no */
       )
    else
      {
        for (i=0; i<20; i++)
          write_string(i+3, 0, table[last+i], NORM_VID);

        /* highlight choice; same relative position on screen */
        write_string(*line_no+3, 0, table[*arrow_choice-20], REV_VID);
      )
    *arrow_choice -= 20;  /* line_no stays the same */
  }
}

/*****************************************************************************
   This function displays next page of table.
     -- if end of table, does nothing;
     -- if last page, moves highlight to last line;
     -- otherwise, highlight stays on same screen line.
*****************************************************************************/
void down_page(arrow_choice,line_no,n,table)
int *arrow_choice;          /* current table entry        */
int *line_no;               /* current screen line number */
int  n;                     /* number of entries in table */
char **table;               /* ship table                 */
{
  int i, num, next;
```

```
num=n-1-*arrow_choice;   /* # lines to final entry from current */

if (*arrow_choice == n-1) ;        /* final table entry */

/* if on last page, but not final entry, highlight last entry */
else if (num < 20)
{
  for (i=0; i<num; i++)
    write_string(i+3, 0, table[n-1-num+i], NORM_VID);

  /* highlight final table entry */
  write_string(num+3, 0, table[n-1], REV_VID);

  *arrow_choice = n-1;
  *line_no = num;

  clear(num+1+3,0,22,79);     /* ensure rest of screen is blank */
}
else   /* not currently on final page */
  {
    /* compute index of top of next page */
    next = *arrow_choice - *line_no + 20;

    /* display next page */
    if (n-next<20)   /* less than 20 lines on final page */
      {
        /* ensure jumping 20 entries won't put highlight past final entry */
        if (next + *line_no < n)       /* it will go past */
          {
            for (i=0; i < (n-next); i++)
              write_string(i+3, 0, table[next+i], NORM_VID);

            /* highlight choice */
            write_string(*line_no+3,0,table[*arrow_choice+20],REV_VID);

            *arrow_choice += 20;       /* line_no stays the same */

            clear(n-next+3,0,22,79);   /* ensure rest of screen is blank */
          }
        else                           /* it won't go past */
          {
            for (i=0; i < (n-next); i++)
              write_string(i+3, 0, table[next+i], NORM_VID);

            /* highlight choice */
            write_string(3, 0, table[next], REV_VID);

            *arrow_choice += 20;       /* line_no stays the same */

            clear(n-next+3,0,22,79);   /* ensure rest of screen is blank */
          )
      )
    else     /* n-next >= 20 */
      {
```

```
          for (i=0; i<20; i++)
            write_string(i+3, 0, table[next+i], NORM_VID);

          /* highlight choice */
          write_string(*line_no+3, 0, table[*arrow_choice+20], REV_VID);

          *arrow_choice += 20;      /* line_no stays the same */
        )
    )
}

/*****************************************************************************
  This function resets display to beginning of table; highlights first entry.
*****************************************************************************/
void goto_home(arrow_choice,line_no,n,table)
int *arrow_choice;           /* current table entry        */
int *line_no;                /* current screen line number */
int  n;                      /* number of entries in table */
char **table;            /* current ship table         */
{
  int i;

  /* display first screen */
  for (i=1; i<20; i++)
    write_string(i+3, 0, table[i], NORM_VID);

  /* highlight first entry */
  write_string(3, 0, table[0], REV_VID);

  /* reset */
  *arrow_choice = 0;
  *line_no = 0;
}

/*****************************************************************************
   This function displays final page of table; highlights final entry.
*****************************************************************************/
void goto_end(arrow_choice,line_no,n,table)
int *arrow_choice;           /* current table entry        */
int *line_no;                /* current screen line number */
int  n;                      /* number of entries in table */
char **table;                /* current ship table         */
{
  int i;

  /* display final page */
  for (i=0; i<19; i++)
    write_string(i+3, 0, table[n-20+i], NORM_VID);

  /* highlight final entry */
  write_string(22, 0, table[n-1], REV_VID);

  *arrow_choice = n-1;    /* set current table entry to final entry */
  *line_no = 19;          /* set current screen line to end of screen */
```

```
}

/*****************************************************************************
     This function reads string data from a file, strips leading and trailing
  spaces, tabs, and carriage returns, fills end of string with blanks, and
  adds a null character to the end of the string.
*****************************************************************************/
void read_table(in,table,length)
FILE *in;          /* pointer to input file              */
char **table;      /* current table                      */
int length;        /* length of desired cursor highlight */
{
  char inline[81];
  int i, j=0;

  while (fgets(inline,81,in) != NULL )
  {
    strstrip(inline);  /* strip leading & trailing spaces, etc. */

    /* skip blank & comment lines */
    if(inline[0] == NULL || inline[0] == '!') continue;

    /* pack end of string with blanks so highlight goes full width */
    for (i=strlen(inline); i<length; i++)
      inline[i] = ' ';

    inline[i] = NULL;    /* terminate string with null character */

    /* allocate enough memory for each string */
    table[j] = (char *) calloc(81,sizeof(char));     /* is this necessary? */
                              /* --array is already dimensioned in main() */

    /* copy string from file into array */
    strcpy(table[j],inline);

    j++;      /* increment counter */
  }
}

/*****************************************************************************
     This function reads strings from a file and converts them to floating
  point data.
*****************************************************************************/
void read_data(in,data_line,data,array_length)
FILE *in;            /* pointer to input file       */
int data_line;       /* ship choice from table      */
float *data;         /* array to store data         */
int array_length;    /* number of elements in array */
{
  char string[81],dummy[81];
  int i,status,n,nbr;

  /* skip blank & comment lines */
  while(1)
```

```
  {
    fgets(string,81,in);
    if(string[0] == NULL || string[0] == '!')
      continue;
    else
      break;
  }

  if (data_line != 0)  /* if choice is first line, string already read */
  {
    for (i=1; i<data_line; i++)  /* read & discard data up to desired line */
      fgets(dummy,81,in);
    fgets(string,81,in);          /* read desired line of data */
  }

  /* convert string to floating array */
  status=stofa(string,data,&nbr,array_length);

  fclose(in);     /* close file */
}

/*****************************************************************************
      This function searches for the matching entry in Table G-2, or gives
  first occurrence of the hull type. If no match is found, returns -1.
*****************************************************************************/
match_ship(hull_no,table,type_match)
char *hull_no;
char **table;
int *type_match;
{
  int i=0, j=0, k=0, n=141;
  int status=0;
  int match=0;
  char class[23];
  int flag=1;

  *type_match = -1;

  do
  {
    status = match_type(hull_no,table[j]);
    if (status)
    {
      if (flag)
        *type_match=j;          /* captures first type match */
      flag=0;
      strncpy(class,table[j],23);
      class[23]=NULL;
      match = match_nbr(hull_no,class);
      if (match)
        break;
      else
      {
        j++;
```

```
        continue;
      }
    }
    else
      j++;
  } while (j<=n);

  if (status)
  {
    return j;
  }
  else
    return -1;
}

/*****************************************************************************
      This function determines if the specified hull type matches that of a
  given entry in Table G-2. Returns 1 if hull type matches, otherwise
  returns 0.
*****************************************************************************/
match_type(hull_no,entry)
char *hull_no;
char *entry;
{
  int i=0;
  int status=0;

  while(hull_no[i] != NULL)
  {
    if (hull_no[i]==' ' && hull_no[i]==entry[i])
    {
      status = 1;
      break;
    }
    else if (hull_no[i]==entry[i])
      i++;
    else
    {
      status = 0;
      break;
    }
  }

  return status;
}

/*****************************************************************************
      This function compares the specified hull number to the list of
  numbers in a given entry in Table G-2. Returns 1 if hull number matches,
  oterwise returns 0.
*****************************************************************************/
match_nbr(hull_no,class)
char *hull_no;
char *class;
```

```
{
  float hull;
  float tab[4];
  int flag[3];
  int k=0, m=0;
  char string[23];
  float nbr;

  /* convert hull_no to actual number */
  while(hull_no[k] != NULL)
  {
    if (hull_no[k] < '0' || hull_no[k] > '9')
      k++;
    else if (hull_no[k] >= '0' && hull_no[k] <= '9')
    {
      string[m]=hull_no[k];
      k++;
      m++;
    }
  }
  string[m]=NULL;
  stofa(string,&hull,&nbr,1);

  /* convert ship class to floating array */
  /*   -- strip off ship type */
  k=0;
  while(class[k] != ' ')
  {
    if (class[k] <= '0' || class[k] >= '9')
      k++;
  }
  /*  -- get first number */
  k++;
  m=0;
  while(class[k] >= '0' && class[k] <= '9')
  {
    string[m]=class[k];
    k++;
    m++;
  }
  if (class[k] == '-')
    flag[0]=0;
  else if (class[k] == ',')
    flag[0]=1;
  else
    flag[0] = -1;

  string[m]=NULL;
  stofa(string,&tab[0],&nbr,1);

  /*  -- get second number */
  k++;
  m=0;
  while(class[k] >= '0' && class[k] <= '9')
```

```
{
  string[m]=class[k];
  k++;
  m++;
}
if (class[k] == '-')
  flag[1]=0;
else if (class[k] == ',')
  flag[1]=1;
else
  flag[1] = -1;

string[m]=NULL;
stofa(string,&tab[1],&nbr,1);

/*  -- get third number */

k++;
m=0;
while(class[k] >= '0' && class[k] <= '9')
{
  string[m]=class[k];
  k++;
  m++;
}
if (class[k] == '-')
  flag[2]=0;
else if (class[k] == ',')
  flag[2]=1;
else
  flag[2] = -1;

string[m]=NULL;
stofa(string,&tab[2],&nbr,1);

/*  -- get fourth number */
k++;
m=0;
while(class[k] >= '0' && class[k] <= '9')
{
  string[m]=class[k];
  k++;
  m++;
}

string[m]=NULL;
stofa(string,&tab[3],&nbr,1);

/* find match */
if (flag[0]<0)        /* only one listed */
{
  if (hull==tab[0])
    return 1;
  else
```

```
    return 0;
)
else if (flag[1]<0)   /* only two listed */
(
  if (flag[0]==1)
  (
    if (hull==tab[0] || hull==tab[1])
      return 1;
    else
      return 0;
  }
  else if (flag[0]==0)
  {
    if (hull>=tab[0] && hull<=tab[1])
      return 1;
    else
      return 0;
  }
}
else if (flag[2]<0)   /* only three listed */
(
  if (flag[0]==1 && flag[1]==1)
  (
    if (hull==tab[0] || hull==tab[1] || hull==tab[2])
      return 1;
    else
      return 0;
  }
  else if (flag[0]==1 && flag[1]==0)
  (
    if (hull==tab[0] || (hull>=tab[1] && hull<=tab[2]) )
      return 1;
    else
      return 0;
  }
  else if (flag[0]==0 && flag[1]==1)
  (
    if ( (hull>=tab[0] && hull<=tab[1]) ||  hull==tab[2])
      return 1;
    else
      return 0;
  }
}
else          /* four listed */
(
  if (flag[0]==1 && flag[1]==1 && flag[2]==1)
  (
    if (hull==tab[0] || hull==tab[1] || hull==tab[2] || hull==tab[3])
      return 1;
    else
      return 0;
  }
  else if (flag[0]==0 && flag[1]==1 && flag[2]==1)
  {
```

```
      if ( (hull>=tab[0] && hull<=tab[1]) || hull==tab[2] || hull==tab[3])
        return 1;
      else
        return 0;
    }
    else if (flag[0]==1 && flag[1]==1 && flag[2]==0)
    {
      if (hull==tab[0] || hull==tab[1] || (hull>=tab[2] && hull<=tab[3]) )
        return 1;
      else
        return 0;
    }
    else if (flag[0]==0 && flag[1]==1 && flag[2]==0)
    {
      if ( (hull>=tab[0] && hull<=tab[1]) || (hull>=tab[2] && hull<=tab[3]) )
        return 1;
      else
        return 0;
    }
  }
}

/*****************************************************************************
      This function reads the ship class based on the user's choice as
  specified by the user in display_tab_g2().
*****************************************************************************/
void read_class(choice,table,class,length)
int choice;
char **table, *class;
int length;
{
  int i;

  for (i=0; i<length; i++}
    class[i] = table[choice][i];

  class[length] = NULL;
}
```

```
/*****************************************************************************
  File: dis.c
  Author: Todd J. Peltzer
  Last update: 30 April 1989

    This file contains the functions which display the ship data read
  from file and give the user the opportunity to edit it.
-----------------------------------------------------------------------------
    Functions:
      display_data()
      edit_data()
*****************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "keydef.h"
#include "video.h"

static char *label[] =
{
  "SHIP DATA",
  "Hull number:",
  "Class:",
  "Displacement",
  "     Actual:",
  "     Tabulated:",
  "Frontal area:",
  "Wind coefficient:",
  "Propeller area:",
  "Hull resistance curve:",
  "Wave resistance curve:"
};

static char *footer[] =
{
  "Is all data correct? (yes/no): ",
  "F1 Disp F2 Front F3 Wind F4 Prop",
  " Press INS to continue ",
  "  Save data to file? (yes/no): ",
  " Enter tug file name: "
};

static char choice[4];

typedef struct
{
  int startcol;
  int endcol;
} data_box;

static data_box input[] =
  {
        {43,50},        /* displacement   */
        {43,50},        /* frontal area   */
        {44,50},        /* wind coef      */
```

```
        {43,50},        /* propeller area */
        {54,58}         /* yes/no choice  */
  };

void edit_data();

/*****************************************************************************
     This function displays the data input by the user to this point, and
  gives the option to edit most items.
*****************************************************************************/
void display_data(hull_no,class,tow_data,ship_data)
char *hull_no;          /* actual hull number        */
char *class;            /* ship class from table G-2 */
float *tow_data;        /* tow data                  */
float *ship_data;       /* ship data                 */
{
  int startrow=2;
  int startcol=20;
  int endrow=22;
  int endcol=60;

  int row, col, i, start, key;
  char string[24];

  /* draw background and border */
  draw_window(startrow,startcol,endrow,endcol,DOUBLE,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,label[0],REV_VID);

  /* write data labels */
  for(row=startrow+5,i=1; i<=10; i++,row++)
    write_string(row,startcol+3,label[i],REV_VID);

  /* write data */
  /* write hull number */
  write_string(startrow+5,startcol+2+strlen(label[1])+2,hull_no,REV_VID);

  /* write ship class */
  write_string(startrow+6,startcol+2+strlen(label[2])+2,class,REV_VID);

  /* write actual displacement */
  sprintf(string,"%8.0f",tow_data[0]);
  write_string(startrow+8,startcol+2+strlen(label[4])+8,string,REV_VID);

  /* write tabulated displacement */
  sprintf(string,"%8.0f",ship_data[0]);
  write_string(startrow+9,startcol+2+strlen(label[5])+5,string,REV_VID);

  /* write frontal windage area */
  sprintf(string,"%5.0f",ship_data[1]);
  write_string(startrow+10,startcol+2+strlen(label[6])+10,string,REV_VID);

  /* write wind coefficient */
```

```
sprintf(string,"%4.2f",ship_data[2]);
write_string(startrow+11,startcol+2+strlen(label[7])+7,string,REV_VID);

/* write propeller area */
sprintf(string,"%5.0f",ship_data[3]);
write_string(startrow+12,startcol+2+strlen(label[8])+8,string,REV_VID);

/* write hull resistance curve number */
sprintf(string,"%3.0f",ship_data[4]);
write_string(startrow+13,startcol+2+strlen(label[9])+3,string,REV_VID);

/* write wave resistance curve number */
sprintf(string,"%3.0f",ship_data[5]);
write_string(startrow+14,startcol+2+strlen(label[10])+3,string,REV_VID);

/* write units */
write_string(startrow+8,startcol+31,"tons",REV_VID);
write_string(startrow+9,startcol+31,"tons",REV_VID);
write_string(startrow+10,startcol+31,"sq ft",REV_VID);
write_string(startrow+12,startcol+31,"sq ft",REV_VID);

/*********************************************************/
/* prompt user for confirmation of data; give edit option */
/*********************************************************/
start = startcol + 3;
write_string(startrow+16,start,footer[0],REV_VID);
vid_box(startrow+16,start+strlen(footer[0]),4);
edit_data(4,startrow+16,start+strlen(footer[0]),tow_data,ship_data);

/* test if data correct */
if ( choice[0]!='y' && choice[0]!='Y' && choice[0]!=NULL )
{
  /* erase previous message */
  start = startcol + 3;
  for (i=0; i<strlen(footer[0])+4; i++)
    write_char(startrow+16,start+i,' ',REV_VID);

  /* write quit message */
  start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
  write_string(endrow,start,footer[2],REV_VID);

  /* write edit "menu" */
  start=(endcol-startcol-strlen(footer[1]))/2 + startcol;
  write_string(startrow+16,start,footer[1],REV_VID);

  /* write function key indicators in NORM_VID */
  for(i=0; i<2; i++)
  {
    write_char(startrow+16,start+i,footer[1][i],NORM_VID);
    write_char(startrow+16,start+8+i,footer[1][i+8],NORM_VID);
    write_char(startrow+16,start+17+i,footer[1][i+17],NORM_VID);
    write_char(startrow+16,start+25+i,footer[1][i+25],NORM_VID);
  }
```

```
/*********************************************************/
/*                    edit data                          */
/*********************************************************/
/* move cursor to F1 highlight */
cursor_on();
goto_xy(startrow+16,start);

/* get user's choice */
while (1)
{
  key=get_special();

  if (key==F1)
  {
    vid_box(startrow+8,startcol+2+strlen(label[4])+9,7);
    edit_data(0,startrow+8,startcol+2+strlen(label[4])+9,tow_data,ship_data);
    for (i=0; i<7; i++)
      write_char(startrow+8,startcol+2+strlen(label[4])+9+i,' ',
                 REV_VID);
    sprintf(string,"%8.0f",tow_data[0]);
    write_string(startrow+8,startcol+2+strlen(label[4])+8,string,
                 REV_VID);
    cursor_on();
    goto_xy(startrow+16,start);
  }
  else if (key==F2)
  {
    vid_box(startrow+10,startcol+2+strlen(label[6])+8,7);
    edit_data(1,startrow+10,startcol+2+strlen(label[6])+8,
            tow_data,ship_data);
    for (i=0; i<7; i++)
      write_char(startrow+10,startcol+2+strlen(label[6])+8+i,' ',
                 REV_VID);
    sprintf(string,"%6.0f",ship_data[1]);
    write_string(startrow+10,startcol+2+strlen(label[6])+9,string,
                 REV_VID);
    cursor_on();
    goto_xy(startrow+16,start+8);
  }
  else if (key==F3)
  {
    vid_box(startrow+11,startcol+2+strlen(label[7])+5,6);
    edit_data(2,startrow+11,startcol+2+strlen(label[7])+5,tow_data,
                 ship_data);
    for (i=0; i<7; i++)
      write_char(startrow+11,startcol+2+strlen(label[7])+5+i,' ',
                 REV_VID);
    sprintf(string,"%6.2f",ship_data[2]);
    write_string(startrow+11,startcol+2+strlen(label[7])+5,string,
                 REV_VID);
    cursor_on();
    goto_xy(startrow+16,start+17);
  }
  else if (key==F4)
```

```
      {
        vid_box(startrow+12,startcol+2+strlen(label[8])+6,7);
        edit_data(3,startrow+12,startcol+2+strlen(label[8])+6,tow_data,
                        ship_data);
        for (i=0; i<7; i++)
          write_char(startrow+12,startcol+2+strlen(label[8])+6+i,' ',
                        REV_VID);
        sprintf(string,"%7.0f",ship_data[3]);
        write_string(startrow+12,startcol+2+strlen(label[8])+6,string,
                        REV_VID);
        cursor_on();
        goto_xy(startrow+16,start+25);
      )
      else if (key==INSERT)
      {
        /* erase menu */
        for (i=0; i<strlen(footer[1]); i++)
          write_char(startrow+16,start+i,' ',REV_VID);

        /* erase message */
        start = startcol + 1;
        for (i=0; i<(endcol-startcol-1); i++)
          write_char(endrow,start+i,205,REV_VID);
        break;
      )
    }
  cursor_off();
  }
  else     /* no editing required */
  {
    /* erase previous message */
    start = startcol + 3;
    for (i=0; i<strlen(footer[0])+4; i++)
      write_char(startrow+16,start+i,' ',REV_VID);
  )
  /* prompt to save data to file */
  start = startcol + 3;
  write_string(endrow-4,start,footer[3],REV_VID);
  vid_box(endrow-4,start+strlen(footer[3]),4);
  edit_data(4,endrow-4,start+strlen(footer[3]),tow_data,ship_data);

  if ( choice[0]!='n' && choice[0]!='N' )     /* save data */
  {
    /* erase previous message */
    start = startcol + 3;
    for (i=0; i<strlen(footer[3])+4; i++)
      write_char(endrow-4,start+i,' ',REV_VID);

    save_ship_file(startrow,startcol,endrow,endcol,hull_no,
                class,tow_data,ship_data);
  }
)

/*****************************************************************************
```

```
   This function allows editing of the data displayed by display_data().
*****************************************************************************/
void edit_data(i,row,col,tow_data,ship_data)
int i;
int row;
int col;
float *tow_data,*ship_data;
{
  int nbr;                    /* required arg for stofa        */
  int status;                 /* takes return value for stofa  */
  char text1[39], text2[39]; /* error message text            */
  int k;                      /* counter                       */
  char string[24];            /* array for keyboard input      */
  cursor_on();                /* turn on cursor                */
  goto_xy(row,col);           /* move cursor to start of box   */

  /* get user input */
  screen_getstrg(i,row,col,string,NORM_VID,input);

  if (i<4)
  {
    /* check for valid numeric input */
    for (k=0; k<strlen(string); k++)
    {
      if ( string[k]=='.' ) ;
      else if (string[k]<'0' || string[k]>'9')
      {
        sprintf(text1,"Invalid entry; try again");
        display_error(ERROR,text1," ");
        vid_box(row,col,7);
        edit_data(i,row,col,tow_data,ship_data);
        return ;
      }
    }
    /* if input valid numbers only, convert string to float */
    if (i==0)
      status = stofa(string,&tow_data[i],&nbr,1);
    else
      status = stofa(string,&ship_data[i],&nbr,1);
  }
  else      /* i == 4 */
    strcpy(choice,string);

  /* check data for consistency */
  if (i==0)     /* check displacement */
  {
    if (tow_data[i]<=0.0)
    {
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
      vid_box(row,col,7);
      edit_data(i,row,col,tow_data,ship_data);
      return ;
    }
```

```
  else if ( tow_data[i]<350.0 || tow_data[i]>91000.0 &&
              tow_data[i]<=100000.0 )
  {
    sprintf(text1,"Displacement is outside range");
    sprintf(text2,"of data in Table G-2");
    display_error(WARN,text1,text2);
  }
  else if ( tow_data[i]>100000.0 )
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    edit_data(i,row,col,tow_data,ship_data);
    return ;
  }
}
if (i==1)    /* check frontal area */
{
  if (ship_data[i]<=0.0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    edit_data(i,row,col,tow_data,ship_data);
    return ;
  }
  else if (ship_data[i]>50000.0)
  {
    sprintf(text1,"Frontal area is unusually high;");
    sprintf(text2,"please check this number again.");
    display_error(WARN,text1,text2);
  }
}
if (i==2)    /* check wind coefficient */
{
  if (ship_data[i]<0 || ship_data[i]>1.0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    edit_data(i,row,col,tow_data,ship_data);
    return ;
  }
}
if (i==3)    /* check propeller area */
{
  if (ship_data[i]<0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    edit_data(i,row,col,tow_data,ship_data);
    return ;
  }
```

```
    else if (ship_data[i]>1500.0)
    {
      sprintf(text1,"Propeller area is unusually high;");
      sprintf(text2,"please check this number again.");
      display_error(WARN,text1,text2);
    }
  }

  cursor_off();
  goto_xy(0,0);
}
```

```
/*****************************************************************************
  File: dock.c
  Author: Todd J. Peltzer
  Last update: 30 April 1989

       This file contains functions which support computation of the
  resistance of floating drydocks and barges.
-----------------------------------------------------------------------------
  Functions:
    get_dock()
    get_hull_cond()
    get_dock_towdata()
    dock_summary()
    get_name()
    get_dock_resist()
    dock_resist()
    square()
    get_dock_data()
    get_dock_file()
    save_dock_file()
    est_disp()
*****************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "keydef.h"
#include "video.h"

static char *header[] =
(
  "SELECT DRYDOCK",
  "HULL CONDITION",
  "TOW DATA",
  "DATA SUMMARY"
);

static char *label[] =
(
  "Clean hull (no growth)          =  0",
  "Average hull (moderate growth) =  5",
  "Fouled hull (heavy growth)      = 10",
  "Enter hull condition:",
  "Tow speed:",
  "Max expected wind speed:",
  "Relative wind direction:"
);

static char *label1[] =
{
  "Options",
  "Retrieve File",
  "Enter name of tow file",
  "to retrieve (8 char max): "
};
```

```
static char *menu1[] =
(
  "1) Enter new data    ",
  "2) Edit existing data",
  "3) Retrieve data file"
};

static char *footer[] =
(
  "Please enter data.",
  "Is all data correct? (yes/no): ",
  "F1 Name F2 Hull F3 Tow",
  "F4 Wind F5 Rel",
  " Press INS to continue ",
  "  Save data to file? (yes/no): ",
  " Enter tow file name: "
};

static char response[4];

typedef struct
{
  int startcol;
  int endcol;
} data_box;

static data_box input[] =
{
        (49,55),        /* not used       */
        {49,55},        /* hull condition */
        {49,55},        /* tow speed      */
        (49,55),        /* wind speed     */
        (49,55},        /* wind direction */
        {54,58},        /* yes/no choice  */
        (49,58)         /* tow file name  */
};

static int startrow=2;  /* boundaries of window */
static int startcol=20; /*          "           */
static int endrow=22;   /*          "           */
static int endcol=60;   /*          "           */

void get_hull_cond(),get_dock_towdata();
void dock_summary(),get_dock_data(),get_name(),get_dock_resist();
void dock_resist(),save_dock_file(),est_disp();

extern float resist_dat[5];

/*****************************************************************************
    This function displays the drydocks listed in Table G-4 of the U.S. Navy
 Towing Manual, gets the user's choice, and reads the appropriate data from
 the table based on that choice.
*****************************************************************************/
get_dock(flag,choice,data,name,tow_data)
```

```
int *flag;          /* status flag */
int *choice;        /* drydock selection  */
float *data;        /* array to hold drydock towing data from Table G-4 */
char *name;         /* drydock name  */
float *tow_data;    /* array to hold tow data */
{
  int start;
  int choice1,status;
  FILE *in;
  char table[15][35];
  char text1[39],text2[39];

  /* clear background */
  clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,header[0],REV_VID);

  if (flag[2])                    /* retrieve data from file */
  {
    status=get_dock_file(startrow,startcol,endrow,endcol,name,
                         data,tow_data);
    if (status<0)
    {
      cursor_off();
      return (-1);
    }
    est_disp(0,data,tow_data);
  }
  else if (!flag[1] && !flag[2])       /* enter new data */
  {
    /* read drydock types from file */
    in = fopen("dock_str.dat","r");
    read_table(in,table,33);
    fclose(in);

    /* use popup function to display drydock choices */
    *choice = popup(table,"",14,startrow+4,startcol+1,NONE,REV_VID,0);

    /* check for ESC key */
    if ( (*choice) < 0)
    {
      status=get_dock(flag,choice,data,name,tow_data);
      return (0);
    }

    /* read drydock name */
    read_class(*choice,table,name,13);

    /* read data from file based on choice */
    in = fopen("drydock.dat","r");
    read_data(in,*choice,data,7);
    fclose(in);
  }
```

```
  return (0);
}

/*****************************************************************************
    This function displays a numerical scale for hull fouling and prompts
 the user for input. Based on that input, the drydock towing coefficient
 "f1" is assigned a value from 0.45 to 0.80.
*****************************************************************************/
void get_hull_cond(tow_data)
float *tow_data;
{
  int row, col, i, key;
  int start;
  char string[81];

  /* draw background and border */
  draw_window(startrow,startcol,endrow,endcol,DOUBLE,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,header[1],REV_VID);

  /* write labels */
  start = (endcol - startcol)/2 + startcol;
  write_string(startrow+5,startcol+3,label[0],REV_VID);
  write_char(startrow+6,start,25,REV_VID);
  write_string(startrow+7,startcol+3,label[1],REV_VID);
  write_char(startrow+8,start,25,REV_VID);
  write_string(startrow+9,startcol+3,label[2],REV_VID);
  write_string(startrow+11,startcol+3,label[3],REV_VID);

  /* create normal video box for data entry */
  vid_box(startrow+11,startcol+3+strlen(label[5])+2,6);

  /* write footer */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  write_string(startrow+13,start,footer[0],REV_VID);

  /*****************************************************************/
  /*                    get hull condition                         */
  /*****************************************************************/
  get_dock_data(4,startrow+11,startcol+3+strlen(label[5])+2,tow_data);
  for (i=0; i<7; i++)
    write_char(startrow+11,startcol+3+strlen(label[5])+2+i,' ',REV_VID);
  sprintf(string,"%7.1f",tow_data[4]);
  write_string(startrow+11,startcol+3+strlen(label[5]),string,REV_VID);

  /* erase prompt */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  for (i=0; i<strlen(footer[0]); i++)
    write_char(startrow+13,start+i,' ',REV_VID);

  /* pause; wait for INS key before continuing */
  pause(startrow,startcol,endrow,endcol,footer[4],REV_VID);
}
```

```
/*****************************************************************
    This function prompts the user for tow speed, wind speed,
  and relative wind direction.
*****************************************************************/
void get_dock_towdata(tow_data)
float *tow_data;
{
  int row, col, i, key;
  int start;
  char string[81];

  /* clear portion of window */
  clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,header[2],REV_VID);

  /* write labels */
  write_string(startrow+7,startcol+3,label[4],REV_VID);
  write_string(startrow+9,startcol+3,label[5],REV_VID);
  write_string(startrow+11,startcol+3,label[6],REV_VID);

  /* create normal video boxes for data entry */
  vid_box(startrow+7,startcol+3+strlen(label[5])+2,6);
  vid_box(startrow+9,startcol+3+strlen(label[5])+2,6);
  vid_box(startrow+11,startcol+3+strlen(label[5])+2,6);

  /* write footer */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  write_string(startrow+13,start,footer[0],REV_VID);

  /* write units */
  write_string(startrow+7,endcol-4,"kts",REV_VID);
  write_string(startrow+9,endcol-4,"kts",REV_VID);
  write_string(startrow+11,endcol-4,"deg",REV_VID);

  /* get data */
  /* -- tow speed */
  get_dock_data(1,startrow+7,startcol+3+strlen(label[5])+2,tow_data);
  for (i=0; i<7; i++)
    write_char(startrow+7,startcol+3+strlen(label[5])+2+i,' ',REV_VID);
  sprintf(string,"%7.1f",tow_data[1]);
  write_string(startrow+7,startcol+2+strlen(label[5])+2,string,REV_VID);

  /* -- wind speed */
  get_dock_data(2,startrow+9,startcol+3+strlen(label[5])+2,tow_data);
  for (i=0; i<7; i++)
    write_char(startrow+9,startcol+3+strlen(label[5])+2+i,' ',REV_VID);
  sprintf(string,"%7.1f",tow_data[2]);
  write_string(startrow+9,startcol+2+strlen(label[5])+2,string,REV_VID);

  /* -- wind dir */
  get_dock_data(3,startrow+11,startcol+3+strlen(label[5])+2,tow_data);
```

```
  for (i=0; i<7; i++)
    write_char(startrow+11,startcol+3+strlen(label[5])+2+i,' ',REV_VID);
  sprintf(string,"%7.1f",tow_data[3]);
  write_string(startrow+11,startcol+2+strlen(label[5])+2,string,REV_VID);

  /* erase prompt */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  for (i=0; i<strlen(footer[0]); i++)
    write_char(startrow+13,start+i,' ',REV_VID);

  /* pause; wait for INS key before continuing */
  pause(startrow,startcol,endrow,endcol,footer[4],REV_VID);
}

/*****************************************************************************
    This function displays a summary of input data and gives the user the
  opportunity to edit.
*****************************************************************************/
void dock_summary(choice,dock_data,tow_data,name)
int *choice;
float *dock_data;
float *tow_data;
char *name;
{
  int row, col, i, key;
  int start;
  char string[81];
  unsigned char *p;  /* buffer for screen data */

  /* clear portion of window */
  clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,header[3],REV_VID);

  /* write labels */
  write_string(startrow+5,startcol+3,"Name:",REV_VID);
  write_string(startrow+7,startcol+3,"Hull condition:",REV_VID);
  write_string(startrow+9,startcol+3,label[4],REV_VID);
  write_string(startrow+11,startcol+3,label[5],REV_VID);
  write_string(startrow+13,startcol+3,label[6],REV_VID);

  /* write data */
  write_string(startrow+5,startcol+3+6,name,REV_VID);
  sprintf(string,"%7.1f",tow_data[4]);
  write_string(startrow+7,startcol+3+strlen(label[5]),string,REV_VID);
  sprintf(string,"%7.1f",tow_data[1]);
  write_string(startrow+9,startcol+3+strlen(label[5]),string,REV_VID);
  sprintf(string,"%7.1f",tow_data[2]);
  write_string(startrow+11,startcol+3+strlen(label[5]),string,REV_VID);
  sprintf(string,"%7.1f",tow_data[3]);
  write_string(startrow+13,startcol+3+strlen(label[5]),string,REV_VID);

  /* write units */
```

```
write_string(startrow+9,startcol+3+strlen(label[5])+9,"kts",REV_VID);
write_string(startrow+11,startcol+3+strlen(label[5])+9,"kts",REV_VID);
write_string(startrow+13,startcol+3+strlen(label[5])+9,"deg",REV_VID);

/* write footer; prompt for confirmation of data */
/* write footer */
write_string(endrow-2,startcol+3,footer[1],REV_VID);

/* create normal video box for data entry */
vid_box(endrow-2,startcol+3+strlen(footer[1]),4);

/* get response */
get_dock_data(5,endrow-2,startcol+3+strlen(footer[1]),tow_data);

/* test if data correct */
if ( response[0]!='y' && response[0]!='Y' && response[0]!=NULL )
{
  /*************************************************************/
  /*                   edit data                               */
  /*************************************************************/
  /* erase previous message */
  start = startcol + 3;
  for (i=0; i<strlen(footer[1])+5; i++)
    write_char(endrow-2,start+i,' ',REV_VID);

  /* write quit message */
  start=(endcol-startcol-strlen(footer[4]))/2 + startcol;
  write_string(endrow,start,footer[4],REV_VID);

  /* write edit "menu" */
  start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
  write_string(startrow+15,start,footer[2],REV_VID);
  start=(endcol-startcol-strlen(footer[3]))/2 + startcol;
  write_string(startrow+16,start,footer[3],REV_VID);

  /* write function key indicators in NORM_VID */
  for(i=0; i<2; i++)
  {
    start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
    write_char(startrow+15,start+i,footer[2][i],NORM_VID);
    write_char(startrow+15,start+8+i,footer[2][i+8],NORM_VID);
    write_char(startrow+15,start+16+i,footer[2][i+16],NORM_VID);
    start=(endcol-startcol-strlen(footer[3]))/2 + startcol;
    write_char(startrow+16,start+i,footer[3][i],NORM_VID);
    write_char(startrow+16,start+8+i,footer[3][i+8],NORM_VID);
  }

  /* move cursor to F1 highlight */
  cursor_on();
  start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
  goto_xy(startrow+15,start);

  /* get user's choice */
  while (1)
```

```
{
  key=get_special();
                        /*****************/
  if (key==F1)          /* drydock name */
  {                     /*****************/
    cursor_off();
    /* allocate enough memory for menu screen buffer*/
    p = (unsigned char *) malloc(2*(endrow-startrow+1)*(endcol-startcol+1));
    if(!p) exit(1); /* install error handler here */

    /* save the current screen data */
    save_screen(startrow,startcol,endrow,endcol,p);

    get_dock(choice,dock_data,name);

    /* restore the original screen*/
    restore_screen(startrow,startcol,endrow,endcol,p);
    free(p);
    write_string(startrow+5,startcol+3+6,name,REV_VID);

    cursor_on();
    start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
    goto_xy(startrow+15,start);
  }                     /*******************/
  else if (key==F2)     /* hull condition */
  {                     /*******************/
    vid_box(startrow+7,startcol+3+strlen(label[5])+2,6);
    get_dock_data(0,startrow+7,startcol+3+strlen(label[5])+2,tow_data);
    for (i=0; i<7; i++)
      write_char(startrow+7,startcol+3+strlen(label[5])+1+i,' ',REV_VID);
    sprintf(string,"%7.1f",tow_data[0]);
    write_string(startrow+7,startcol+3+strlen(label[5]),string,REV_VID);
    cursor_on();
    start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
    goto_xy(startrow+15,start+8);
  }                     /*************/
  else if (key==F3)     /* tow speed */
  {                     /*************/
    vid_box(startrow+9,startcol+3+strlen(label[5])+2,6);
    get_dock_data(1,startrow+9,startcol+3+strlen(label[5])+2,tow_data);
    for (i=0; i<7; i++)
      write_char(startrow+9,startcol+3+strlen(label[5])+1+i,' ',REV_VID);
    sprintf(string,"%7.1f",tow_data[1]);
    write_string(startrow+9,startcol+3+strlen(label[5]),string,REV_VID);
    cursor_on();
    start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
    goto_xy(startrow+15,start+16);
  }                     /**************/
  else if (key==F4)     /* wind speed */
  {                     /**************/
    vid_box(startrow+11,startcol+3+strlen(label[5])+2,6);
    get_dock_data(2,startrow+11,startcol+3+strlen(label[5])+2,tow_data);
    for (i=0; i<7; i++)
      write_char(startrow+11,startcol+3+strlen(label[5])+1+i,' ',REV_VID);
```

```
      sprintf(string,"%7.1f",tow_data[2]);
      write_string(startrow+11,startcol+3+strlen(label[5]),string,REV_VID);
      cursor_on();
      start=(endcol-startcol-strlen(footer[3]))/2 + startcol;
      goto_xy(startrow+16,start);
    )                    /******************/
    else if (key==F5)    /* wind direction */
    {                    /******************/
      vid_box(startrow+13,startcol+3+strlen(label[5])+2,6);
      get_dock_data(3,startrow+13,startcol+3+strlen(label[5])+2,tow_data);
      for (i=0; i<7; i++)
        write_char(startrow+13,startcol+3+strlen(label[5])+1+i,' ',REV_VID);
      sprintf(string,"%7.1f",tow_data[3]);
      write_string(startrow+13,startcol+3+strlen(label[5]),string,REV_VID);
      cursor_on();
      start=(endcol-startcol-strlen(footer[3]))/2 + startcol;
      goto_xy(startrow+16,start+8);
    )
    else if (key==INSERT)
    {
      /* erase menu */
      start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
      for (i=0; i<strlen(footer[2])+1; i++)
      {
        write_char(startrow+15,start+i,' ',REV_VID);
        write_char(startrow+16,start+i,' ',REV_VID);
      }
      /* erase message */
      start = (endcol-startcol-strlen(footer[4]))/2 + startcol;
      for (i=0; i<strlen(footer[4]); i++)
        write_char(endrow,start+i,205,REV_VID);
      break;
    )
  }
)
else     /* no editing required */
{
  /* erase previous message */
  start = startcol + 3;
  for (i=0; i<strlen(footer[0])+4; i++)
    write_char(endrow-2,start+i,' ',REV_VID);
)
/* prompt to save data to file */
start = startcol + 3;
write_string(endrow-2,start,footer[5],REV_VID);
vid_box(endrow-2,start+strlen(footer[5]),4);
get_dock_data(5,endrow-2,start+strlen(footer[5]),tow_data);

if ( response[0]!='n' && response[0]!='N' )    /* save data */
{
  /* erase previous message */
  start = startcol + 3;
  for (i=0; i<strlen(footer[5])+4; i++)
    write_char(endrow-2,start+i,' ',REV_VID);
```

```
    save_dock_file(startrow,startcol,endrow,endcol,name,
                dock_data,tow_data);
  }

}

/*****************************************************************************
     This function computes the towing resistance of a drydock based
  on the method in section G-2 of the U.S. Navy Towing Manual.
*****************************************************************************/
float square();

void get_dock_resist(name,tug_data,dock_data,tow_data,tension)
char *name;
float *tug_data;
float *dock_data;
float *tow_data;
float *tension;
{
  static char header[] =
  {
    "RESISTANCE"
  };

  static char *label[] =
  {
    "Drydock:",
    "Table G-4 data",
    "  f1:",
    "  f2:",
    "  f3:",
    "  Wetted surface area:",
    "  Cross sectional area",
    "    Below waterline:",
    "    Above waterline:",
    "Resistance",
    "  Frictional:",
    "  Wave forming:",
    "  Wind:",
    "  Hawser:",
    "           Total:"
  };

  int row, col, i, key;
  int start;
  char string[81];

  float friction;
  float wave;
  float wind;
  float f1;
  float resistance;
  float haw_res;
```

```
cursor_off();

/* convert hull condition to f1 coefficient */
f1 = tow_data[4]/10.0 * (0.8 - 0.45) + 0.45;

/* compute resistance */
friction = f1 * dock_data[0] * square(tow_data[1]/6.0);
wave = 2.85*dock_data[1]*dock_data[2]*square(tow_data[1])*1.2;
wind = dock_data[4]*0.004*square(tow_data[1]+tow_data[2])*dock_data[3];
resistance = friction + wave + wind;

/* save data for report */
resist_dat[0]=wind;
wave_height(tow_data[2],&resist_dat[1]);
resist_dat[2]=wave;
resist_dat[3]=friction;
resist_dat[4]=0.0;            /* no propeller resistance */

/* compute hawser resistance */
hawser_resist(tug_data,tow_data[1],resistance,&haw_res);

/* compute total resistance */
*tension = resistance + haw_res;

/*************************************************************/
/*                   display results                         */
/*************************************************************/
/* draw background and border */
draw_window(startrow,startcol,endrow,endcol,DOUBLE,REV_VID);

/* write header */
write_header(startrow,startcol,endcol,header,REV_VID);

/* write labels */
for (i=0; i<15; i++)
  write_string(startrow+4+i,startcol+3,label[i],REV_VID);

/************************/
/*    write data        */
/************************/

/* drydock name */
write_string(startrow+4,startcol+3+strlen(label[0])+1,name,REV_VID);

sprintf(string,"%7.2f",f1);              /* f1 */
write_string(startrow+6,startcol+3+strlen(label[5]),string,REV_VID);

sprintf(string,"%7.2f",dock_data[1]);    /* f2 */
write_string(startrow+7,startcol+3+strlen(label[5]),string,REV_VID);

sprintf(string,"%7.2f",dock_data[3]);    /* f3 */
write_string(startrow+8,startcol+3+strlen(label[5]),string,REV_VID);
```

```
  sprintf(string,"%7.0f",dock_data[0]);   /* wetted surface area */
  write_string(startrow+9,startcol+3+strlen(label[5]),string,REV_VID);

  sprintf(string,"%7.0f",dock_data[2]);   /* below waterline cross sect */
  write_string(startrow+11,startcol+3+strlen(label[5]),string,REV_VID);

  sprintf(string,"%7.0f",dock_data[4]);   /* above waterline cross sect */
  write_string(startrow+12,startcol+3+strlen(label[5]),string,REV_VID);

  sprintf(string,"%7.0f",friction);       /* frictional resistance */
  write_string(startrow+14,startcol+3+strlen(label[5]),string,REV_VID);

  sprintf(string,"%7.0f",wave);           /* wave-forming resistance */
  write_string(startrow+15,startcol+3+strlen(label[5]),string,REV_VID);

  sprintf(string,"%7.0f",wind);           /* wind resistance */
  write_string(startrow+16,startcol+3+strlen(label[5]),string,REV_VID);

  sprintf(string,"%7.0f",haw_res);          /* hawser resistance */
  write_string(startrow+17,startcol+3+strlen(label[5]),string,REV_VID);

  sprintf(string,"%7.0f",resistance);    /* total resistance */
  write_string(startrow+18,startcol+3+strlen(label[5]),string,REV_VID);

  /* write units */
  write_string(startrow+9,endcol-6,"sq ft",REV_VID);
  write_string(startrow+11,endcol-6,"sq ft",REV_VID);
  write_string(startrow+12,endcol-6,"sq ft",REV_VID);
  for (i=0; i<5; i++)
    write_string(startrow+14+i,endcol-6,"lbs",REV_VID);

  /* pause; wait for INS key before continuing */
  pause(startrow,startcol,endrow,endcol,footer[4],REV_VID);

}

/*****************************************************************
     This function computes the square of a number.
*****************************************************************/
float square(x)
float x;
{
  return (x*x);
}

/*****************************************************************
     This function gets the user's input for the data requested
  in the function get_hull_cond(). Uses screen_getstring() for
  screen I/O. Some data checking is performed for each data type.
*****************************************************************/
void get_dock_data(i,row,col,data)
int i;
int row;
int col;
```

```
float *data;
{
  int nbr;                    /* required arg for stofa       */
  int status;                 /* takes return value for stofa */
  char string[24];            /* input string                 */
  char text1[39], text2[39]; /* error message text           */
  int j, k;                   /* counters                     */

  cursor_on();                /* turn on cursor               */
  goto_xy(row,col);           /* move cursor to start of box  */

  /* get user input */
  screen_getstrg(i,row,col,string,NORM_VID,input);

  if (i != 5)
  {
    /* check for valid numeric input */
    for (k=0; k<strlen(string); k++)
    {
      if ( string[k]=='.' )
      {
        if (string[k+1]=='.')
        {
          sprintf(text1,"Invalid entry; try again");
          display_error(ERROR,text1," ");
          vid_box(row,col,7);
          get_dock_data(i,row,col,data);
          return ;
        }
      }
      else if (string[k]<'0' || string[k]>'9')
      {
        sprintf(text1,"Invalid entry; try again");
        display_error(ERROR,text1," ");
        vid_box(row,col,7);
        get_dock_data(i,row,col,data);
        return ;
      }
    }
    /* if input valid numbers only, convert string to float */
    status = stofa(string,&data[i],&nbr,1);
  }
  else if (i==5)
  {
    strcpy(response,string);
  }

  /* check data for consistency */
  if (i==1)    /* check towing speed */
  {
    if (data[i]<=0.0 || data[i]>12.0)
    {
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
```

```
    vid_box(row,col,6);
    get_dock_data(i,row,col,data);
    return ;
  )
  else if (data[i]>10.0 && data[i]<=12.0)
  (
    sprintf(text1,"Planned tow speed is higher than");
    sprintf(text2,"normally recommended");
    display_error(WARN,text1,text2);
  )
)
if (i==2)    /* check wind speed */
(
  if (data[i]<0)
  (
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,6);
    get_dock_data(i,row,col,data);
    return ;
  )
  else if (data[i]>48.0)       /* ??? is this true with P-M spectrum ??? */
  (
    sprintf(text1,"Outside range of Fig. G-2");
    sprintf(text2,"=>please enter new wind speed<=");
    display_error(ERROR,text1,text2);
    vid_box(row,col,6);
    get_dock_data(i,row,col,data);
    return ;
  )
)
if (i==3)    /* check wind direction */
(
  if (data[i]<0)
  {
    sprintf(text1,"Use only positive angles");
    display_error(WARN,text1," ");
    data[i] = -data[i];
  )
  else if (data[i]>180.0 && data[i]<360.0)
  (
    sprintf(text1,"Use angles between 0 and 180");
    display_error(WARN,text1," ");
    data[i] = 360.0 - data[i];
  )
  else if (data[i]>70.0 && data[i]<110.0)
  (
    sprintf(text1,"No extreme tension data for beam seas");
    sprintf(text2,"==> use angles < 70 or > 110 <==");
    display_error(ERROR,text1,text2);
    vid_box(row,col,6);
    get_dock_data(i,row,col,data);
    return ;
  )
```

```
    else if (data[i]>360.0)
    {
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
      vid_box(row,col,6);
      get_dock_data(i,row,col,data);
      return ;
    )
  }
  if (i==4)    /* check hull condition */
  (
    if (data[i]<0.0 || data[i]>10.0)
    {
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
      vid_box(row,col,6);
      get_dock_data(i,row,col,data);
      return ;
    }
  }

  cursor_off();
  goto_xy(0,0);
)

/*************************************************************************
     This function estimates the displacement of floating drydocks
  and barges based on the assumptions:
        Cb = 0.8
        Cx = 0.9
*************************************************************************/
void est_disp(i,data,tow_data)
int i;
float *data;
float *tow_data;
{
  float length,beam,draft;
  float coefb=0.8;              /* block coefficient */
  float coefx=0.9;              /* max section coefficient */

  if (i==0)    /* drydock */
  (
    length=data[5];
    beam=data[6];
    draft = data[2]/(coefx*beam);
    tow_data[0] = length*beam*draft*coefb/35.0;
  }
  else         /* barge */
  (
    length=data[0];
    beam=data[1];
    draft=data[3];
    tow_data[0] = length*beam*draft*coefb/35.0;
  )
```

```
}

/*****************************************************************
    This function computes the mean tension of a floating
  drydock and its towing hawser. It duplicates the computations
  in the function get_dock_resist(), but is intended as a stand
  alone function for use by other program modules.
*****************************************************************/
void dock_resist(tug_data,dock_data,tow_data,tension)
float *tug_data;
float *dock_data;
float *tow_data;
float *tension;
{
  float f1;
  float friction;
  float wave;
  float wind;
  float resistance;
  float haw_res;

  /* convert hull condition to f1 coefficient */
  f1 = tow_data[4]/10.0 * (0.8 - 0.45) + 0.45;

  /* compute resistance */
  friction = f1 * dock_data[0] * square(tow_data[1]/6.0);
  wave = 2.85*dock_data[1]*dock_data[2]*square(tow_data[1])*1.2;
  wind = dock_data[4]*0.004*square(tow_data[1]+tow_data[2])*dock_data[3];
  resistance = friction + wave + wind;

  /* compute hawser resistance */
  hawser_resist(tug_data,tow_data[1],resistance,&haw_res);

  /* compute total resistance */
  *tension = resistance + haw_res;
}


/*****************************************************************
  This function retrieves tow data from a user specified file.
*****************************************************************/
get_dock_file(startrow,startcol,endrow,endcol,hull_no,
                dock_data,tow_data)
int startrow,startcol;
int endrow,endcol;
char *hull_no;
float *dock_data;
float *tow_data;
{
  FILE *in;
  char fname[13];
  char inline[81];
  char text1[39],text2[39];
  int i,j,status,nbr;
```

```
float array[2];
int start;

/* write header */
start = (endcol-startcol-strlen(label1[1]))/2 + startcol;
write_string(startrow+8,start,label1[1],REV_VID);

/* write prompt */
for (i=0; i<2; i++)
{
  start = startcol+3;
  write_string(startrow+10+i,start,label1[i+2],REV_VID);
}
vid_box(startrow+11,start+strlen(label1[3]),9);
sprintf(text1,"Type 'Q' to quit");
start = (endcol-startcol-strlen(text1))/2 + startcol;
write_string(endrow-3,start,text1,REV_VID);
get_fname(6,1,startrow+11,startcol+3+strlen(label1[3]),fname);

/* check for quit option */
if (fname[0]=='Q')
{
  cursor_off();
  goto_xy(0,0);
  return (-1);
}

in=fopen(fname,"r");
if (in==NULL)
{
  sprintf(text1,"Can't open file %s",fname);
  display_error(ERROR,text1," ");
  clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
  status=get_dock_file(startrow,startcol,endrow,endcol,hull_no,
              dock_data,tow_data);
  return (0);
}
else
{
  /* read data from file */
  fgets(inline,81,in);
  strstrip(inline);

  /* test for valid tow file */
  if (!(strncmp(inline,"! Tow data file",15)))
  {
    fgets(inline,81,in);
    fgets(inline,81,in);
    strstrip(inline);

    /* test if drydock file */
    if (strncmp(inline,"DRYDOCK",7))
    {
      sprintf(text1,"%s is not a",fname);
```

```
        sprintf(text2,"drydock data file!");
        display_error(ERROR,text1,text2);
        clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
        status=get_dock_file(startrow,startcol,endrow,endcol,hull_no,
                dock_data,tow_data);
        return (0);
      )

      /* get hull number */
      fgets(inline,81,in);
      fgets(inline,81,in);
      strstrip(inline);
      strncpy(hull_no,inline,24);
      strstrip(hull_no);

      /* read remaining tow data */
      for (i=0; i<5; i++)
      {
        fgets(inline,81,in);
        fgets(inline,81,in);
        status=stofa(inline,array,&nbr,1);
        tow_data[i]=array[0];
      }

      /* read drydock data */
      for (i=0; i<7; i++)
      (
        fgets(inline,81,in);
        fgets(inline,81,in);
        status=stofa(inline,array,&nbr,1);
        dock_data[i]=array[0];
      )
      fclose(in);
    }
    else
    {
      sprintf(text1,"%s is not a tow data file!",fname);
      display_error(ERROR,text1," ");
      clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
      status=get_dock_file(startrow,startcol,endrow,endcol,hull_no,
                dock_data,tow_data);
      return (0);
    }
  )
  return (0);
)

/*****************************************************************
  This function writes the tow data for floating drydocks to
  a user specified file.
*****************************************************************/
void save_dock_file(startrow,startcol,endrow,endcol,hull_no,
                dock_data,tow_data)
int startrow,startcol,endrow,endcol;
```

```
char *hull_no;
float *dock_data;
float *tow_data;
(
  FILE *out;
  char fname[13];
  char string[25];
  char text1[39];
  int start;

  /* prompt for file name */
  start = startcol + 3;
  write_string(endrow-2,start,footer[6],REV_VID);
  vid_box(endrow-2,start+strlen(footer[6])+4,9);

  get_fname(6,1,endrow-2,start+strlen(footer[6])+4,fname);
  out=fopen(fname,"w");

  if (out==NULL)
  (
    sprintf(text1,"Can't open file %s",fname);
    display_error(ERROR,text1," ");
    save_dock_file(startrow,startcol,endrow,endcol,hull_no,
                dock_data,tow_data);
    return;
  }
  else
  (
    /* write tow data to file */
    fprintf(out,"! Tow data file\n");
    fprintf(out,"! Tow type:\n");
    fprintf(out,"DRYDOCK\n");

    fprintf(out,"! Hull number:\n");
    fprintf(out,"%s\n",hull_no);

    fprintf(out,"! Estimated displacement:\n");
    sprintf(string,"%-.1f\n",tow_data[0]);
    fprintf(out,"%s",string);

    fprintf(out,"! Tow speed (kts):\n");
    sprintf(string,"%-.1f\n",tow_data[1]);
    fprintf(out,"%s",string);

    fprintf(out,"! Wind speed (kts):\n");
    sprintf(string,"%-.2f\n",tow_data[2]);
    fprintf(out,"%s",string);

    fprintf(out,"! Relative wind dir:\n");
    sprintf(string,"%-.2f\n",tow_data[3]);
    fprintf(out,"%s",string);

    fprintf(out,"! Hull condition:\n");
    sprintf(string,"%-.2f\n",tow_data[4]);
```

```
    fprintf(out,"%s",string);

    /* write drydock data to file */
    fprintf(out,"! Wetted surface area:\n");
    sprintf(string,"%-.0f\n",dock_data[0]);
    fprintf(out,"%s",string);

    fprintf(out,"! Form factor (f2):\n");
    sprintf(string,"%-.2f\n",dock_data[1]);
    fprintf(out,"%s",string);

    fprintf(out,"! Cross-sectional area below waterline (B):\n");
    sprintf(string,"%-.2f\n",dock_data[2]);
    fprintf(out,"%s",string);

    fprintf(out,"! Wind coefficient (f3):\n");
    sprintf(string,"%-.2f\n",dock_data[3]);
    fprintf(out,"%s",string);

    fprintf(out,"! Cross-sectional area above waterline (C):\n");
    sprintf(string,"%-.0f\n",dock_data[4]);
    fprintf(out,"%s",string);

    fprintf(out,"! Drydock length:\n");
    sprintf(string,"%-.1f\n",dock_data[5]);
    fprintf(out,"%s",string);

    fprintf(out,"! Drydock beam:\n");
    sprintf(string,"%-.1f\n",dock_data[6]);
    fprintf(out,"%s",string);
  }
  fclose(out);
}
```

```
/*****************************************************************************
  File: barge.c
  Author: Todd J. Peltzer
  Last update: 3 May 1989

       This file contains the functions which support computation of the
  resistance of floating drydocks and barges.
-----------------------------------------------------------------------------
  Functions:
    get_barge_resist()
    barge_resist()
    get_barge_data()
    barge_summary()
    save_barge_file()
    get_barge_file()
*****************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "keydef.h"
#include "video.h"

#define RAKE 0
#define SHIPSHAPE 1
#define SQUARE 2

void get_barge_data(), barge_summary();
void barge_resist(),save_barge_file();
float square();

typedef struct
{
  int startcol;
  int endcol;
} data_box;

static data_box input[] =
{
  {42,55},        /* name (dimension) */
  {40,53},        /* name (summary)   */
  {46,52},        /* hull length      */
  {46,52},        /* beam             */
  {46,52},        /* hull depth       */
  {46,52},        /* draft            */
  {46,52},        /* deckhouse length */
  {46,52},        /* deckhouse width  */
  {46,52},        /* deckhouse height */
  {54,58}         /* yes/no choice    */
};

static char response[4];

static char *footer[] =
{
  "Please enter data.",
```

```
  "Is all data correct? (yes/no): ",
  "F1 Name F2 Hull F3 Tow",
  "F4 Wind F5 Rel",
  " Press INS to continue ",
  "  Save data to file? (yes/no): ",
  " Enter tow file name: "
};

static char *label[] =
{
  "Name or hull no:",
  "Hull dimensions",
  "  Length:",
  "  Beam:",
  "  Depth:",
  "  Draft:",
  "Deckhouse dimensions",
  "  Length:",
  "  Width:",
  "  Height:",
  "End shape:"
};

static char *label1[] =
{
  "Name or hull no:",
  "Data corresponding to Table G-4",
  "  f1:",
  "  f2:",
  "  f3:",
  "  Wetted surface area:",
  "  Cross sectional area",
  "    Below waterline:",
  "    Above waterline:",
  "Resistance",
  "  Frictional:",
  "  Wave_forming:",
  "  Wind:",
  "  Hawser:",
  "           Total:"
};

static char *menu[] =
{
  "Rake ended  ",
  "Ship ended  ",
  "Square ended"
};

static char *label2[] =
{
  "Options",
  "Retrieve File",
  "Enter name of tow file",
```

```
  "to retrieve (8 char max): "
};

static char *menu1[] =
{
  "1) Enter new data    ",
  "2) Edit existing data",
  "3) Retrieve data file"
};

static int startrow=2;
static int startcol=20;
static int endrow=22;
static int endcol=60;

static char *header[] =
{
  "BARGE DATA",
  "DATA SUMMARY",
  "RESISTANCE"
};

extern float barge_res[6],resist_dat[5];     /* arrays for report data */

/*****************************************************************************
     This function prompts the user for the dimensions of the barge hull
  and deckhouse (and allows editing), then prompts for the hull condition,
  the tow speed and wind speed, prints a summary of input data (and allows
  editing), then computes the towing resistance of the barge and prints
  a summary.
*****************************************************************************/
get_barge_resist(flag,name,tug_data,tow_data,barge_data,tension)
int *flag;
char *name;
float *tug_data;
float *tow_data;
float *barge_data;          /* hull dim (4), dkhs dim (3), end shape */
float *tension;
{
  int row, col, i, key;
  int start,choice1,status;
  char string[81];
  char text1[39],text2[39];
  float vtow;               /* tow speed  */
  float vwind;              /* wind speed */
  float f1,f2,f3;           /* resistance coefficients */
  float wet_surf;           /* wetted surface area */
  float uw_xsect;           /* underwater cross-sectional area */
  float abv_xsect;          /* abovewater cross-sectional area */
  float hlength;            /* hull length */
  float beam;               /* hull beam   */
  float depth;              /* hull depth  */
  float draft;              /* full load draft  */
  float dlength;            /* deckhouse length */
```

```
float width;                /* deckhouse width  */
float height;               /* deckhouse height */
int end_shape;
float friction;
float wave_resist;
float wind_resist;
float resistance;
float haw_res;

if (!flag[1])                   /* barge data does not exist */
{
 /* initialize data */
 for (i=0; i<8; i++)
   barge_data[i] = 0.0;
}

/ clear background */
clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

/ write header */
write_header(startrow,startcol,endcol,header[0],REV_VID);

if (!flag[1] && !flag[2])         /* enter new data */
{
  lear(startrow+5,startcol+1,endrow-3,endcol-1,REV_VID);
}
else if (flag[1] && !flag[2]) ;  /* edit existing data */
else if (flag[2])     /* retrieve data from file */
{
  tatus=get_barge_file(name,barge_data,tow_data);
  f (status<0)

   cursor_off();
   flag[2]=0;
   return (-1);

  st_disp(1,barge_data,tow_data);
}
clear(startrow+5,startcol+1,endrow-3,endcol-1,REV_VID);

/*write labels */
fo (i=0; i<10; i++)
  rite_string(startrow+5+i,startcol+5,label[i],REV_VID);
write_string(startrow+16,startcol+5,label[10],REV_VID);

/*write units */
fo (i=0; i<4; i++)
  rite_string(startrow+7+i,endcol-5,"ft",REV_VID);
fo (i=0; i<3; i++)
  rite_string(startrow+12+i,endcol-5,"ft",REV_VID);

if(flag[1] || flag[2])          /* display data */
{
  rite_string(startrow+5,startcol+5+strlen(label[0])+2,name,REV_VID);
```

```
  start=startcol+5+strlen(label[6])+1;

  for (i=0; i<4; i++)
  {
    sprintf(string,"%7.1f",barge_data[i]);
    write_string(startrow+7+i,start,string,REV_VID);
  }
  for (i=0; i<3; i++)
  {
    sprintf(string,"%7.1f",barge_data[i+4]);
    write_string(startrow+12+i,start,string,REV_VID);
  }
  if (barge_data[7] == RAKE)
  {
    write_string(startrow+16,start+1,menu[0],REV_VID);
  }
  else if (barge_data[7] == SHIPSHAPE)
  {
    write_string(startrow+16,start+1,menu[1],REV_VID);
  }
  else if (barge_data[7] == SQUARE)
  {
    write_string(startrow+16,start+1,menu[2],REV_VID);
  }
}
else if (!flag[1] && !flag[2])           /* enter new data */
{
  /* write footer */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  write_string(endcol-2,start,footer[0],REV_VID);

  /*****************************************/
  /*                get data                */
  /*****************************************/
  /* get name */
  vid_box(startrow+5,startcol+5+strlen(label[0])+1,14);
  get_barge_data(0,startrow+5,startcol+5+strlen(label[0])+1,name,barge_data);
  for (i=0; i<15; i++)
    write_char(startrow+5,startcol+5+strlen(label[0])+1+i,' ',REV_VID);
  write_string(startrow+5,startcol+5+strlen(label[0])+2,name,REV_VID);

  start=startcol+5+strlen(label[6])+1;

  /* get hull length */
  vid_box(startrow+7,start,7);
  get_barge_data(2,startrow+7,start,name,barge_data);
  for (i=0; i<7; i++)
    write_char(startrow+7,start+i,' ',REV_VID);
  sprintf(string,"%7.1f",barge_data[0]);
  write_string(startrow+7,start,string,REV_VID);

  /* get beam */
  vid_box(startrow+8,start,7);
```

```
get_barge_data(3,startrow+8,start,name,barge_data);
for (i=0; i<7; i++)
  write_char(startrow+8,start+i,' ',REV_VID);
sprintf(string,"%7.1f",barge_data[1]);
write_string(startrow+8,start,string,REV_VID);

/* get hull depth */
vid_box(startrow+9,start,7);
get_barge_data(4,startrow+9,start,name,barge_data);
for (i=0; i<7; i++)
  write_char(startrow+9,start+i,' ',REV_VID);
sprintf(string,"%7.1f",barge_data[2]);
write_string(startrow+9,start,string,REV_VID);

/* get draft */
vid_box(startrow+10,start,7);
get_barge_data(5,startrow+10,start,name,barge_data);
for (i=0; i<7; i++)
  write_char(startrow+10,start+i,' ',REV_VID);
sprintf(string,"%7.1f",barge_data[3]);
write_string(startrow+10,start,string,REV_VID);

/* get deckhouse length */
vid_box(startrow+12,start,7);
get_barge_data(6,startrow+12,start,name,barge_data);
for (i=0; i<7; i++)
  write_char(startrow+12,start+i,' ',REV_VID);
sprintf(string,"%7.1f",barge_data[4]);
write_string(startrow+12,start,string,REV_VID);

/* get deckhouse width */
vid_box(startrow+13,start,7);
get_barge_data(7,startrow+13,start,name,barge_data);
for (i=0; i<7; i++)
  write_char(startrow+13,start+i,' ',REV_VID);
sprintf(string,"%7.1f",barge_data[5]);
write_string(startrow+13,start,string,REV_VID);

/* get deckhouse height */
vid_box(startrow+14,start,7);
get_barge_data(8,startrow+14,start,name,barge_data);
for (i=0; i<7; i++)
  write_char(startrow+14,start+i,' ',REV_VID);
sprintf(string,"%7.1f",barge_data[6]);
write_string(startrow+14,start,string,REV_VID);

/* get end shape */
start = startcol+5+strlen(label[8])+7;
barge_data[7] = popup(menu,"",3,startrow+15,start,SINGLE,REV_VID,0);

if (barge_data[7] == RAKE)
{
  write_string(startrow+16,start+1,menu[0],REV_VID);
}
```

```
  else if (barge_data[7] == SHIPSHAPE)
  {
    write_string(startrow+16,start+1,menu[1],REV_VID);
  }
  else if (barge_data[7] == SQUARE)
  {
    write_string(startrow+16,start+1,menu[2],REV_VID);
  }
}

/*****************************************************************/
/*    prompt user for confirmation of data; give edit option     */
/*****************************************************************/
start = startcol+3;
write_string(endrow-2,start,footer[1],REV_VID);
vid_box(endrow-2,start+strlen(footer[1]),4);
get_barge_data(9,endrow-2,start+strlen(footer[1]),name,barge_data);

/* test if data correct */
if ( response[0]!='y' && response[0]!='Y' && response[0]!=NULL )
{
  /* erase previous message */
  start = startcol+3;
  for (i=0; i<strlen(footer[1])+4; i++)
    write_char(endrow-2,start+i,' ',REV_VID);

  /* write quit message */
  start=(endcol-startcol-strlen(footer[4]))/2 + startcol;
  write_string(endrow,start,footer[4],REV_VID);

  /* write function key indicators in NORM_VID */
  write_string(startrow+5,startcol+2,"F1",NORM_VID);
  write_string(startrow+7,startcol+2,"F2",NORM_VID);
  write_string(startrow+8,startcol+2,"F3",NORM_VID);
  write_string(startrow+9,startcol+2,"F4",NORM_VID);
  write_string(startrow+10,startcol+2,"F5",NORM_VID);
  write_string(startrow+12,startcol+2,"F6",NORM_VID);
  write_string(startrow+13,startcol+2,"F7",NORM_VID);
  write_string(startrow+14,startcol+2,"F8",NORM_VID);
  write_string(startrow+16,startcol+2,"F9",NORM_VID);

  /* move cursor to F1 highlight */
  cursor_on();
  goto_xy(startrow+5,startcol+2);

  /* get user's choice */
  while (1)
  {
    start=startcol+5+strlen(label[6])+1;
    key=get_special();
                  /***********************/
    if (key==F1)  /* hull name or number */
    {             /***********************/
      vid_box(startrow+5,startcol+5+strlen(label[0])+1,14);
```

```
  get_barge_data(0,startrow+5,startcol+5+strlen(label[0])+1,name,barge_data);
  for (i=0; i<15; i++)
    write_char(startrow+5,startcol+5+strlen(label[0])+1+i,' ',REV_VID);
  write_string(startrow+5,startcol+5+strlen(label[0])+2,name,REV_VID);
  cursor_on();
  goto_xy(startrow+5,startcol+2);
}                    /***************/
else if (key==F2)    /* hull length */
{                    /***************/
  vid_box(startrow+7,start,7);
  get_barge_data(2,startrow+7,start,name,barge_data);
  for (i=0; i<7; i++)
    write_char(startrow+7,start+i,' ',REV_VID);
  sprintf(string,"%7.1f",barge_data[0]);
  write_string(startrow+7,start,string,REV_VID);
  cursor_on();
  goto_xy(startrow+7,startcol+2);

  /* check deckhouse length for consistency with this input */
  if (barge_data[4]>barge_data[0])   /* is greater than hull length */
  {
    sprintf(text1,"Deckhouse is longer than hull");
    sprintf(text2,"==> please try again <==");
    display_error(ERROR,text1,text2);
    vid_box(startrow+12,start,7);
    get_barge_data(6,startrow+12,start,name,barge_data);
    for (i=0; i<7; i++)
      write_char(startrow+12,start+i,' ',REV_VID);
    sprintf(string,"%7.1f",barge_data[4]);
    write_string(startrow+12,start,string,REV_VID);
    cursor_on();
    goto_xy(startrow+12,startcol+2);
  }
}                     /********/
else if (key==F3)     /* beam */
{                     /********/
  vid_box(startrow+8,start,7);
  get_barge_data(3,startrow+8,start,name,barge_data);
  for (i=0; i<7; i++)
    write_char(startrow+8,start+i,' ',REV_VID);
  sprintf(string,"%7.1f",barge_data[1]);
  write_string(startrow+8,start,string,REV_VID);
  cursor_on();
  goto_xy(startrow+8,startcol+2);

  /* check deckhouse width for consistency with this input */
  if (barge_data[5]>barge_data[1])   /* is greater than beam */
  {
    sprintf(text1,"Deckhouse is wider than beam");
    sprintf(text2,"==> please try again <==");
    display_error(ERROR,text1,text2);
    vid_box(startrow+13,start,7);
    get_barge_data(7,startrow+13,start,name,barge_data);
    for (i=0; i<7; i++)
```

```
      write_char(startrow+13,start+i,' ',REV_VID);
    sprintf(string,"%7.1f",barge_data[5]);
    write_string(startrow+13,start,string,REV_VID);
    cursor_on();
    goto_xy(startrow+13,startcol+2);
  }
}                       /**************/
else if (key==F4)       /* hull depth */
{                       /**************/
  vid_box(startrow+9,start,7);
  get_barge_data(4,startrow+9,start,name,barge_data);
  for (i=0; i<7; i++)
    write_char(startrow+9,start+i,' ',REV_VID);
  sprintf(string,"%7.1f",barge_data[2]);
  write_string(startrow+9,start,string,REV_VID);
  cursor_on();
  goto_xy(startrow+9,startcol+2);
}                       /*********/
else if (key==F5)       /* draft */
{                       /*********/
  vid_box(startrow+10,start,7);
  get_barge_data(5,startrow+10,start,name,barge_data);
  for (i=0; i<7; i++)
    write_char(startrow+10,start+i,' ',REV_VID);
  sprintf(string,"%7.1f",barge_data[3]);
  write_string(startrow+10,start,string,REV_VID);
  cursor_on();
  goto_xy(startrow+10,startcol+2);
}                      /********************/
else if (key==F6)      /* deckhouse length */
{                      /********************/
  vid_box(startrow+12,start,7);
  get_barge_data(6,startrow+12,start,name,barge_data);
  for (i=0; i<7; i++)
    write_char(startrow+12,start+i,' ',REV_VID);
  sprintf(string,"%7.1f",barge_data[4]);
  write_string(startrow+12,start,string,REV_VID);
  cursor_on();
  goto_xy(startrow+12,startcol+2);
}                       /********************/
else if (key==F7)       /* deckhouse width */
{                       /********************/
  vid_box(startrow+13,start,7);
  get_barge_data(7,startrow+13,start,name,barge_data);
  for (i=0; i<7; i++)
    write_char(startrow+13,start+i,' ',REV_VID);
  sprintf(string,"%7.1f",barge_data[5]);
  write_string(startrow+13,start,string,REV_VID);
  cursor_on();
  goto_xy(startrow+13,startcol+2);
}                       /********************/
else if (key==F8)       /* deckhouse height */
{                       /********************/
  vid_box(startrow+14,start,7);
```

```
      get_barge_data(8,startrow+14,start,name,barge_data);
      for (i=0; i<7; i++)
        write_char(startrow+14,start+i,' ',REV_VID);
      sprintf(string,"%7.1f",barge_data[6]);
      write_string(startrow+14,start,string,REV_VID);
      cursor_on();
      goto_xy(startrow+14,startcol+2);
    }                       /*************/
    else if (key==F9)       /* end shape */
    {                       /*************/
      start = startcol+5+strlen(label[8])+7;
      barge_data[7] = popup(menu,"",3,startrow+15,start,SINGLE,REV_VID,0);
      if (barge_data[7] == RAKE)
      {
        write_string(startrow+16,start+1,menu[0],REV_VID);
      }
      else if (barge_data[7] == SHIPSHAPE)
      {
        write_string(startrow+16,start+1,menu[1],REV_VID);
      }
      else if (barge_data[7] == SQUARE)
      {
        write_string(startrow+16,start+1,menu[2],REV_VID);
      }
      cursor_on();
      goto_xy(startrow+16,startcol+2);
    }                           /*************/
    else if (key==INSERT)       /* quit edit */
    {                           /*************/
      /* erase function key highlights */
      write_string(startrow+5,startcol+2,"  ",REV_VID);
      write_string(startrow+7,startcol+2,"  ",REV_VID);
      write_string(startrow+8,startcol+2,"  ",REV_VID);
      write_string(startrow+9,startcol+2,"  ",REV_VID);
      write_string(startrow+10,startcol+2,"  ",REV_VID);
      write_string(startrow+12,startcol+2,"  ",REV_VID);
      write_string(startrow+13,startcol+2,"  ",REV_VID);
      write_string(startrow+14,startcol+2,"  ",REV_VID);
      write_string(startrow+16,startcol+2,"  ",REV_VID);

      /* erase quit message */
      start = (endcol-startcol-strlen(footer[4]))/2 + startcol;
      for (i=0; i<strlen(footer[4]); i++)
        write_char(endrow,start+i,205,REV_VID);
      break;
    }
  }
cursor_off();
}
else            /* editing not required */
{
  /* erase previous message */
  start = startcol + 4;
  for (i=0; i<strlen(footer[1])+4; i++)
```

```
    write_char(endrow-2,start+i,' ',REV_VID);
}

if (!flag[1] && !flag[2])
{
  /*****************************************/
  /*     get hull condition and tow data;    */
  /*        estimate displacement            */
  /*****************************************/
  /* use same functions as used for drydocks */
  get_hull_cond(tow_data);

  get_dock_towdata(tow_data);

  est_disp(1,barge_data,tow_data);
}

/*****************************************************************/
/*               display summary; give edit option               */
/*****************************************************************/
barge_summary(barge_data,tow_data,name);

/*****************************************************************/
/*               compute barge resistance                        */
/*****************************************************************/
vtow = tow_data[1];
vwind = tow_data[2];
hlength = barge_data[0];
beam = barge_data[1];
depth = barge_data[2];
draft = barge_data[3];
dlength = barge_data[4];
width = barge_data[5];
height = barge_data[6];
end_shape = barge_data[7];

/* determine f1, f2, f3 */
f1 = (tow_data[4]/10.0)*(0.8 - 0.45) + 0.45;   /* hull condition */
barge_res[0]=f1;
if (end_shape==0 || end_shape==1)
{
  f2 = 0.2;    /* rake or ship-ended */
}
else
{
  f2 = 0.5;    /* square-ended */
}
barge_res[1]=f2;
f3 = 0.60;     /* average barge value per Section G-3.2 */
barge_res[2]=f3;

/* compute frictional resistance */
wet_surf = (hlength*beam) + 2.0*(hlength+beam)*draft;
barge_res[3]=wet_surf;
```

```
vtow = square(vtow/6.0);
friction = f1 * wet_surf * vtow;
resist_dat[3]=friction;

/* compute wave resistance */
uw_xsect = (beam*draft);
barge_res[4]=uw_xsect;
wave_resist = 2.85 * uw_xsect * f2 * square(vtow) * 1.2;
resist_dat[2]=wave_resist;
wave_height(vwind,&resist_dat[1]);

/* compute wind resistance */
abv_xsect = (depth-draft)*beam + height*width;
barge_res[5]=abv_xsect;
wind_resist = abv_xsect*0.004*square(vtow+vwind)*f3;
resist_dat[0]=wind_resist;

/* compute total resistance */
resistance = friction + wave_resist + wind_resist;
resist_dat[4]=0;              /* no propeller resistance */

/* compute hawser resistance */
hawser_resist(tug_data,tow_data[1],resistance,&haw_res);

/* compute total resistance */
*tension = resistance + haw_res;

/*************************************************************/
/*                  display results                          */
/*************************************************************/
/* clear portion of window */
clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

/* write header */
write_header(startrow,startcol,endcol,header[2],REV_VID);

/* write labels */
for (i=0; i<15; i++)
  write_string(startrow+4+i,startcol+3,label1[i],REV_VID);

/************************/
/*    write data        */
/************************/

/* barge name */
write_string(startrow+4,startcol+3+strlen(label1[0])+1,name,REV_VID);

sprintf(string,"%7.2f",f1);           /* f1 */
write_string(startrow+6,startcol+3+strlen(label1[5]),string,REV_VID);

sprintf(string,"%7.2f",f2);           /* f2 */
write_string(startrow+7,startcol+3+strlen(label1[5]),string,REV_VID);

sprintf(string,"%7.2f",f3);           /* f3 */
```

```
  write_string(startrow+8,startcol+3+strlen(label1[5]),string,REV_VID);

  sprintf(string,"%7.0f",wet_surf);     /* wetted surface area */
  write_string(startrow+9,startcol+3+strlen(label1[5]),string,REV_VID);

  sprintf(string,"%7.0f",uw_xsect);     /* below waterline cross sect */
  write_string(startrow+11,startcol+3+strlen(label1[5]),string,REV_VID);

  sprintf(string,"%7.0f",abv_xsect);   /* above waterline cross sect */
  write_string(startrow+12,startcol+3+strlen(label1[5]),string,REV_VID);

  sprintf(string,"%7.0f",friction);       /* frictional resistance */
  write_string(startrow+14,startcol+3+strlen(label1[5]),string,REV_VID);

  sprintf(string,"%7.0f",wave_resist);    /* wave-forming resistance */
  write_string(startrow+15,startcol+3+strlen(label1[5]),string,REV_VID);

  sprintf(string,"%7.0f",wind_resist);    /* wind resistance */
  write_string(startrow+16,startcol+3+strlen(label1[5]),string,REV_VID);

  sprintf(string,"%7.0f",haw_res);       /* hawser resistance */
  write_string(startrow+17,startcol+3+strlen(label1[5]),string,REV_VID);

  sprintf(string,"%7.0f",*tension);      /* total resistance */
  write_string(startrow+18,startcol+3+strlen(label1[5]),string,REV_VID);

  /* write units */
  write_string(startrow+9,endcol-6,"sq ft",REV_VID);
  write_string(startrow+11,endcol-6,"sq ft",REV_VID);
  write_string(startrow+12,endcol-6,"sq ft",REV_VID);
  for (i=0; i<5; i++)
    write_string(startrow+14+i,endcol-6,"lbs",REV_VID);

  /* pause; wait for INS key before continuing */
  pause(startrow,startcol,endrow,endcol,footer[4],REV_VID);

  return (0);
}

/*************************************************************************
     This function gets the user's input for the data requested in the
  function get_barge_resist(). Uses screen_getstrg() for screen I/O. Some
  data checking is performed for each data type.
*************************************************************************/
void get_barge_data(i,row,col,name,data)
int i;
int row;
int col;
char *name;
float *data;
{
  int nbr;                  /* required arg for stofa       */
  int status;               /* takes return value for stofa */
  char string[13];          /* input string                 */
```

```
char text1[39], text2[39]; /* error message text           */
int j, k;                  /* counters                      */
int length;

cursor_on();               /* turn on cursor                */
goto_xy(row,col);          /* move cursor to start of box   */

/* get user input */
screen_getstrg(i,row,col,string,NORM_VID,input);

if (i==0 || i==1)   /* barge name or hull number */
{
  /* convert input string to upper case; output to name */
  slctouc(string,name);

  /* ensure hull number is in proper format */
  /* -- first copy name */
  strcpy(string,name);

  /* -- ensure space between hull type and number */
  length=strlen(name);
  for (j=0; j<length; j++)
  {
    if (name[j]>='0' && name[j]<='9')
    {
      if (name[j-1] == '-')
      {
        name[j-1] = ' ';
        break;
      }
      else if (name[j-1] != ' ')
      {
        string[j] = ' ';
        for (k=j+1; k<=strlen(name); k++)
          string[k] = name[k-1];
        string[k] = NULL;
        slctouc(string,name);
        break;
      }
      else
      {
        break;
      }
    }
  }
}
else if (i>1 && i<9)
{
  /* check for valid numeric input */
  for (k=0; k<strlen(string); k++)
  {
    if ( string[k]=='.' ) ;
    else if (string[k]<'0' || string[k]>'9')
    {
```

```
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
      vid_box(row,col,7);
      get_barge_data(i,row,col,name,data);
      return ;
    }
  )
  /* if input string contains only valid numbers, convert string to float */
  status = stofa(string,&data[i-2],&nbr,1);
}
else      /* i == 9; edit response */
{
  strcpy(response,string);
)

/*******************************************************/
/*            check data for consistency               */
/*******************************************************/
               /***********************/
if (i<2)       /* hull name or number */
{              /***********************/
  if (name[0]<'A' || 'Z'<name[0] && name[0]<'a' || name[0]>'z')
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,14);
    get_barge_data(i,row,col,name,data);
    return ;
  }
}              /***************/
if (i==2)      /* hull length */
{              /***************/
  if (data[0]<=0.0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  )
  else if ( data[0]>1000.0 )
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  }
}              /********/
if (i==3)      /* beam */
{              /********/
  if (data[1]<=0.0)
  {
    sprintf(text1,"Invalid entry; try again");
```

```
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  }
  else if ( data[1]>500.0 )
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  }
}           /***************/
if (i==4)   /* hull depth */
{           /***************/
  if (data[2]<=0.0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  }
  else if ( data[2]>100.0 )
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  }
}           /**********/
if (i==5)   /* draft */
{           /**********/
  if (data[3]<=0.0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  }
  else if ( data[3]>data[2] )
  {
    sprintf(text1,"Draft is deeper than hull");
    sprintf(text2,"==> please try again <==");
    display_error(ERROR,text1,text2);
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  }
}           /********************/
if (i==6)   /* deckhouse length */
```

```
{            /*********************/
  if (data[4]<0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  )
  else if (data[4]>data[0])   /* is greater than hull length */
  {
    sprintf(text1,"Deckhouse is longer than hull");
    sprintf(text2,"==> please try again <==");
    display_error(ERROR,text1,text2);
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  }
)            /*******************/
if (i==7)    /* deckhouse width */
{            /*******************/
  if (data[5]<0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  )
  else if (data[5]>data[1])   /* is greater than beam */
  {
    sprintf(text1,"Deckhouse is wider than beam");
    sprintf(text2,"==> please try again <==");
    display_error(ERROR,text1,text2);
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  )
}            /*********************/
if (i==8)    /* deckhouse height */
{            /*********************/
  if (data[6]<0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_barge_data(i,row,col,name,data);
    return ;
  )
  else if ( data[6]>100.0 )
  {
    sprintf(text1,"Deckhouse is unusually high");
    sprintf(text2,"==> please try again <==");
    display_error(ERROR,text1,text2);
```

```
      vid_box(row,col,7);
      get_barge_data(i,row,col,name,data);
      return ;
    }
  }

  cursor_off();
  goto_xy(0,0);
}

/*****************************************************************************
    This function displays a summary of input data and gives the user the
  opportunity to edit.
*****************************************************************************/
void barge_summary(barge_data,tow_data,name)
float *barge_data;
float *tow_data;
char *name;
{
  static char *label1[] =
  {
    "Name or hull no:",
    "Hull condition:",
    "Tow speed:",
    "Max expected wind speed:",
    "Relative wind direction:"
  };

  int row, col, i, key;
  int start;
  char string[81],string1[81];

  /* clear portion of window */
  clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,header[1],REV_VID);

  /* write labels */
  write_string(startrow+5,startcol+3,label1[0],REV_VID);
  write_string(startrow+7,startcol+3,label1[1],REV_VID);
  write_string(startrow+9,startcol+3,label1[2],REV_VID);
  write_string(startrow+11,startcol+3,label1[3],REV_VID);
  write_string(startrow+13,startcol+3,label1[4],REV_VID);

  /* write data */
  write_string(startrow+5,startcol+3+strlen(label1[0])+2,name,REV_VID);
  sprintf(string,"%7.1f",tow_data[4]);
  write_string(startrow+7,startcol+3+strlen(label1[3]),string,REV_VID);
  sprintf(string,"%7.1f",tow_data[1]);
  write_string(startrow+9,startcol+3+strlen(label1[3]),string,REV_VID);
  sprintf(string,"%7.1f",tow_data[2]);
  write_string(startrow+11,startcol+3+strlen(label1[3]),string,REV_VID);
  sprintf(string,"%7.1f",tow_data[3]);
```

```
write_string(startrow+13,startcol+3+strlen(label1[3]),string,REV_VID);

/* write units */
write_string(startrow+9,endcol-4,"kts",REV_VID);
write_string(startrow+11,endcol-4,"kts",REV_VID);
write_string(startrow+13,endcol-4,"deg",REV_VID);

/* write footer; prompt for confirmation of data */
/* write footer */
write_string(endrow-2,startcol+3,footer[1],REV_VID);

/* create normal video box for response */
vid_box(endrow-2,startcol+3+strlen(footer[1]),4);

/* get response */
get_barge_data(9,endrow-2,startcol+3+strlen(footer[1]),name,tow_data);

/* test if data correct */
if ( response[0]!='y' && response[0]!='Y' && response[0]!=NULL )
{
  /*************************************************************/
  /*                   edit data                               */
  /*************************************************************/
  /* erase previous message */
  start = startcol + 3;
  for (i=0; i<strlen(footer[1])+5; i++)
    write_char(endrow-2,start+i,' ',REV_VID);

  /* write quit message */
  start=(endcol-startcol-strlen(footer[4]))/2 + startcol;
  write_string(endrow,start,footer[4],REV_VID);

  /* write edit "menu" */
  start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
  write_string(startrow+15,start,footer[2],REV_VID);
  start=(endcol-startcol-strlen(footer[3]))/2 + startcol;
  write_string(startrow+16,start,footer[3],REV_VID);

  /* write function key indicators in NORM_VID */
  for(i=0; i<2; i++)
  {
    start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
    write_char(startrow+15,start+i,footer[2][i],NORM_VID);
    write_char(startrow+15,start+8+i,footer[2][i+8],NORM_VID);
    write_char(startrow+15,start+16+i,footer[2][i+16],NORM_VID);
    start=(endcol-startcol-strlen(footer[3]))/2 + startcol;
    write_char(startrow+16,start+i,footer[3][i],NORM_VID);
    write_char(startrow+16,start+8+i,footer[3][i+8],NORM_VID);
  }

  /* move cursor to F1 highlight */
  cursor_on();
  start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
  goto_xy(startrow+15,start);
```

```
/* get user's choice */
while (1)
{
  key=get_special();
                        /***************/
  if (key==F1)          /* barge name */
  {                     /***************/
    vid_box(startrow+5,startcol+3+strlen(label1[0])+2,14);
    get_barge_data(1,startrow+5,startcol+3+strlen(label1[0])+2,name,barge_data);
    for (i=0; i<=15; i++)
      write_char(startrow+5,startcol+3+strlen(label1[0])+1+i,' ',REV_VID);
    write_string(startrow+5,startcol+3+strlen(label1[0])+2,name,REV_VID);
    cursor_on();
    start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
    goto_xy(startrow+15,start);
  }                     /*******************/
  else if (key==F2)     /* hull condition */
  {                     /*******************/
    vid_box(startrow+7,startcol+3+strlen(label1[3])+2,6);
    get_dock_data(4,startrow+7,startcol+3+strlen(label1[3])+2,tow_data);
    for (i=0; i<7; i++)
      write_char(startrow+7,startcol+3+strlen(label1[3])+1+i,' ',REV_VID);
    sprintf(string,"%7.1f",tow_data[4]);
    write_string(startrow+7,startcol+3+strlen(label1[3]),string,REV_VID);
    cursor_on();
    start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
    goto_xy(startrow+15,start+8);
  }                     /**************/
  else if (key==F3)     /* tow speed */
  {                     /**************/
    vid_box(startrow+9,startcol+3+strlen(label1[3])+2,6);
    get_dock_data(1,startrow+9,startcol+3+strlen(label1[3])+2,tow_data);
    for (i=0; i<7; i++)
      write_char(startrow+9,startcol+3+strlen(label1[3])+1+i,' ',REV_VID);
    sprintf(string,"%7.1f",tow_data[1]);
    write_string(startrow+9,startcol+3+strlen(label1[3]),string,REV_VID);
    cursor_on();
    start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
    goto_xy(startrow+15,start+16);
  }                     /***************/
  else if (key==F4)     /* wind speed */
  {                     /***************/
    vid_box(startrow+11,startcol+3+strlen(label1[3])+2,6);
    get_dock_data(2,startrow+11,startcol+3+strlen(label1[3])+2,tow_data);
    for (i=0; i<7; i++)
      write_char(startrow+11,startcol+3+strlen(label1[3])+1+i,' ',REV_VID);
    sprintf(string,"%7.1f",tow_data[2]);
    write_string(startrow+11,startcol+3+strlen(label1[3]),string,REV_VID);
    cursor_on();
    start=(endcol-startcol-strlen(footer[3]))/2 + startcol;
    goto_xy(startrow+16,start);
  }                     /***********************/
  else if (key==F5)     /* rel wind direction */
```

```
    {                    /**********************/
      vid_box(startrow+13,startcol+3+strlen(label1[3])+2,6);
      get_dock_data(3,startrow+13,startcol+3+strlen(label1[3])+2,tow_data);
      for (i=0; i<7; i++)
        write_char(startrow+13,startcol+3+strlen(label1[3])+1+i,' ',REV_VID);
      sprintf(string,"%7.1f",tow_data[3]);
      write_string(startrow+13,startcol+3+strlen(label1[3]),string,REV_VID);
      cursor_on();
      start=(endcol-startcol-strlen(footer[3]))/2 + startcol;
      goto_xy(startrow+16,start+8);
    }
    else if (key==INSERT)
    {
      /* erase menu */
      for (i=0; i<strlen(footer[2]); i++)
      {
        start=(endcol-startcol-strlen(footer[2]))/2 + startcol;
        write_char(startrow+15,start+i,' ',REV_VID);
        write_char(startrow+16,start+i,' ',REV_VID);
      }
      /* erase message */
      start = (endcol-startcol-strlen(footer[4]))/2 + startcol;
      for (i=0; i<strlen(footer[4]); i++)
        write_char(endrow,start+i,205,REV_VID);
      cursor_off();
      goto_xy(0,0);
      break;
    }
  }
 }
 /* prompt to save data to file */
 /* -- erase message */
 start = (endcol-startcol-strlen(footer[1]))/2 + startcol;
 for (i=0; i<37; i++)
   write_char(startrow+15,startcol+3+i,' ',REV_VID);

 start = startcol + 3;
 write_string(endrow-2,start,footer[5],REV_VID);
 vid_box(endrow-2,start+strlen(footer[5]),4);
 get_barge_data(9,endrow-2,start+strlen(footer[5]),name,tow_data);

 if ( response[0]!='n' && response[0]!='N' )     /* save data */
 {
   /* erase previous message */
   start = startcol + 3;
   for (i=0; i<strlen(footer[5])+4; i++)
     write_char(endrow-2,start+i,' ',REV_VID);

   save_barge_file(name,barge_data,tow_data);
 }
}

/*****************************************************************
    This function computes the mean tension of a barge and its
```

```
  towing hawser. It duplicates the computations in the function
  get_barge_resist(), but is intended as a stand alone function
  for use by other program modules.
*****************************************************************/
void barge_resist(tug_data,barge_data,tow_data,tension)
float *tug_data;
float *barge_data;
float *tow_data;
float *tension;
{
  float vtow = tow_data[1];
  float vwind = tow_data[2];
  float hlength = barge_data[0];
  float beam = barge_data[1];
  float depth = barge_data[2];
  float draft = barge_data[3];
  float dlength = barge_data[4];
  float width = barge_data[5];
  float height = barge_data[6];
  float end_shape = barge_data[7];
  float f1,f2,f3;
  float wet_surf,friction,uw_xsect,abv_xsect;
  float wave_resist,wind_resist,resistance,haw_res;

  /* determine f1, f2, f3 */
  f1 = (tow_data[4]/10.0)*(0.8 - 0.45) + 0.45;   /* hull condition */

  if (end_shape==0 || end_shape==1)
    f2 = 0.2;     /* rake or ship-ended */
  else
    f2 = 0.5;     /* square-ended */

  f3 = 0.60;      /* average barge value per Section G-3.2 */

  /* compute frictional resistance */
  wet_surf = (hlength*beam) + 2.0*(hlength+beam)*draft;
  vtow = square(vtow/6.0);
  friction = f1 * wet_surf * vtow;

  /* compute wave resistance */
  uw_xsect = (beam*draft);
  wave_resist = 2.85 * uw_xsect * f2 * square(vtow) * 1.2;

  /* compute wind resistance */
  abv_xsect = (depth-draft)*beam + height*width;
  wind_resist = abv_xsect*0.004*square(vtow+vwind)*f3;

  /* compute total resistance */
  resistance = friction + wave_resist + wind_resist;

  /* compute hawser resistance */
  hawser_resist(tug_data,tow_data[1],resistance,&haw_res);

  /* compute total resistance */
```

```
  *tension = resistance + haw_res;
}

/*****************************************************************
  This function retrieves tow data from a user specified file.
*****************************************************************/
get_barge_file(hull_no,barge_data,tow_data)
char *hull_no;
float *barge_data;
float *tow_data;
{
  FILE *in;
  char fname[13];
  char inline[81];
  char text1[39],text2[39];
  int i,j,status,nbr;
  float array[2];
  int start;

  /* write header */
  start = (endcol-startcol-strlen(label2[1]))/2 + startcol;
  write_string(startrow+8,start,label2[1],REV_VID);

  /* write prompt */
  for (i=0; i<2; i++)
  {
    start = startcol+3;
    write_string(startrow+10+i,start,label2[i+2],REV_VID);
  }
  vid_box(startrow+11,start+strlen(label2[3]),9);
  sprintf(text1,"Type 'Q' to quit");
  start = (endcol-startcol-strlen(text1))/2 + startcol;
  write_string(endrow-3,start,text1,REV_VID);
  get_fname(6,1,startrow+11,startcol+3+strlen(label2[3]),fname);

  /* check for quit option */
  if (fname[0]=='Q')
  {
    cursor_off();
    goto_xy(0,0);
    return (-1);
  }

  in=fopen(fname,"r");
  if (in==NULL)
  {
    sprintf(text1,"Can't open file %s",fname);
    display_error(ERROR,text1," ");
    clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
    status=get_barge_file(hull_no,barge_data,tow_data);
    if (status<0)
    {
      cursor_off();
      goto_xy(0,0);
```

```
    return (-1);
  )
  else
    return (0);
)
else
{
  /* read data from file */
  fgets(inline,81,in);
  strstrip(inline);

  /* test for valid tow file */
  if (!(strncmp(inline,"! Tow data file",15)))
  {
    fgets(inline,81,in);
    fgets(inline,81,in);
    strstrip(inline);

    /* test if barge file */
    if (strncmp(inline,"BARGE",5))
    {
      sprintf(text1,"%s is not a",fname);
      sprintf(text2,"barge data file!");
      display_error(ERROR,text1,text2);
      clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
      status=get_barge_file(hull_no,barge_data,tow_data);
      if (status<0)
      {
        cursor_off();
        goto_xy(0,0);
        return (-1);
      }
      else
        return (0);
    }

    /* get hull number */
    fgets(inline,81,in);
    fgets(inline,81,in);
    strstrip(inline);
    strncpy(hull_no,inline,24);
    strstrip(hull_no);

    /* read remaining tow data */
    for (i=0; i<5; i++)
    {
      fgets(inline,81,in);
      fgets(inline,81,in);
      status=stofa(inline,array,&nbr,1);
      tow_data[i]=array[0];
    }

    /* read barge data */
    for (i=0; i<8; i++)
```

```
      (
        fgets(inline,81,in);
        fgets(inline,81,in);
        status=stofa(inline,array,&nbr,1);
        barge_data[i]=array[0];
      }
      fclose(in);
    }
    else
    (
      sprintf(text1,"%s is not a tow data file!",fname);
      display_error(ERROR,text1," ");
      clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
      status=get_barge_file(hull_no,barge_data,tow_data);
      if (status<0)
      {
        cursor_off();
        goto_xy(0,0);
        return (-1);
      }
      else
        return (0);
    }
  }
}

/*****************************************************************
  This function writes the tow data for barges to a user
  specified file.
*****************************************************************/
void save_barge_file(hull_no,barge_data,tow_data)
char *hull_no;
float *barge_data;
float *tow_data;
{
  FILE *out;
  char fname[13];
  char string[25];
  char text1[39],text2[39];
  int start;

  /* prompt for file name */
  start = startcol + 3;
  write_string(endrow-2,start,footer[6],REV_VID);
  vid_box(endrow-2,start+strlen(footer[6])+4,9);

  get_fname(6,1,endrow-2,start+strlen(footer[6])+4,fname);
  out=fopen(fname,"w");

  if (out==NULL)
  (
    sprintf(text1,"Can't open file %s",fname);
    display_error(ERROR,text1," ");
    save_barge_file(hull_no,barge_data,tow_data);
```

```
  return;
}
else
{
  /* write tow data to file */
  fprintf(out,"! Tow data file\n");
  fprintf(out,"! Tow type:\n");
  fprintf(out,"BARGE\n");

  fprintf(out,"! Hull number:\n");
  fprintf(out,"%s\n",hull_no);

  fprintf(out,"! Estimated displacement:\n");
  sprintf(string,"%-.1f\n",tow_data[0]);
  fprintf(out,"%s",string);

  fprintf(out,"! Tow speed (kts):\n");
  sprintf(string,"%-.1f\n",tow_data[1]);
  fprintf(out,"%s",string);

  fprintf(out,"! Wind speed (kts):\n");
  sprintf(string,"%-.2f\n",tow_data[2]);
  fprintf(out,"%s",string);

  fprintf(out,"! Relative wind dir:\n");
  sprintf(string,"%-.2f\n",tow_data[3]);
  fprintf(out,"%s",string);

  fprintf(out,"! Hull condition:\n");
  sprintf(string,"%-.2f\n",tow_data[4]);
  fprintf(out,"%s",string);

  /* write barge data to file */
  fprintf(out,"! Hull length:\n");
  sprintf(string,"%-.1f\n",barge_data[0]);
  fprintf(out,"%s",string);

  fprintf(out,"! Beam:\n");
  sprintf(string,"%-.1f\n",barge_data[1]);
  fprintf(out,"%s",string);

  fprintf(out,"! Hull depth:\n");
  sprintf(string,"%-.1f\n",barge_data[2]);
  fprintf(out,"%s",string);

  fprintf(out,"! Draft:\n");
  sprintf(string,"%-.1f\n",barge_data[3]);
  fprintf(out,"%s",string);

  fprintf(out,"! Deckhouse length:\n");
  sprintf(string,"%-.1f\n",barge_data[4]);
  fprintf(out,"%s",string);

  fprintf(out,"! Deckhouse width:\n");
```

```
    sprintf(string,"%-.1f\n",barge_data[5]);
    fprintf(out,"%s",string);

    fprintf(out,"! Deckhouse height:\n");
    sprintf(string,"%-.1f\n",barge_data[6]);
    fprintf(out,"%s",string);

    fprintf(out,"! End shape:\n");
    sprintf(string,"%-.0f\n",barge_data[7]);
    fprintf(out,"%s",string);
  }
  fclose(out);
}
```

```
/*************************************************************************
  File: drg.c
  Author: Todd J. Peltzer
  Last update: 30 April 1989

    This file contains the functions which compute the tow resistance
  of self-propelled ships, as well as the resistance of the tow
  hawser.
-------------------------------------------------------------------------
  Functions:
    mean_tension()
    ship_resist()
    added_res()
    hull_resist()
    wave_height()
    hawser_resist()
*************************************************************************/
#include "stdio.h"
#include "math.h"
#include "keydef.h"
#include "video.h"

#define PI 3.141592654

static char *label[] =
{
  "Wind resistance:",
  "Wave height:",
  "Wave resistance:",
  "Hull resistance:",
  "Propeller resistance:",
  "Total tow resistance:",
  "Hawser resistance:",
  "Mean tension:"
};

static char footer[] =
{
  " Press INS to continue "
};

static char *header[] =
{
  "RESISTANCE",
  "DATA SUMMARY"
};

static char *label1[] =
{
  "Hull no:",
  "Class:",
  "Disp:",
  "Tab disp:",
  "Front area:",
```

```
  "Wind coef:",
  "Prop area:",
  "Hull curve:",
  "Wave curve:",
  "Tug class:",
  "Hawser",
  "  diameter:",
  "  scope:"
};

static char *label2[] =
{
  "Tow spd:",
  "Max wind:",
  "Rel wind dir:",
  "Wave height:",
  "Prop status:",
  "Chain pendant",
  "  size:",
  "  scope:"
};

static char *label3[] =
{
  "Resistance (lbs)",
  "  Wind:",
  "  Wave:",
  "  Hull:",
  "  Prop:",
  "Total:",
  "Hawser:",
  "Tension:"
};

extern float resist_dat[5];

void mean_tension(),ship_resist(),hawser_resist();
float square();

/*************************************************************************
    This function computes the mean towing tension of a ship, displays
  the results, and then displays a summary.
*************************************************************************/
void mean_tension(tug_data,tow_data,ship_data,hull_no,class,tension)
float *tug_data;        /* tug data    */
float *tow_data;        /* tow data    */
float *ship_data;       /* ship data   */
char *hull_no;          /* hull number */
char *class;            /* ship class  */
float *tension;         /* mean hawser tension */
{
  int row,col,i,start,key;
  int status;
  float tow_res;        /* total tow resistance          */
```

```
float haw_res;          /* hawser resistance              */
float res_dat[5];       /* array of resistances           */
int startrow=2;
int startcol=20;
int endrow=22;
int endcol=60;
int left,center,right;
char string[15];

/* start of computations */
ship_resist(tow_data,ship_data,res_dat,&tow_res);
for (i=0; i<5; i++)
  resist_dat[i]=res_dat[i];   /* save data for report */

/*********************************/
/* compute total tow resistance */
/*********************************/

/* compute hawser resistance */
hawser_resist(tug_data,tow_data[1],tow_res,&haw_res);

/* compute mean tension */
*tension = tow_res+haw_res;

/************************************************/
/*                display results               */
/************************************************/
/* draw background and border */
draw_window(startrow,startcol,endrow,endcol,DOUBLE,REV_VID);

/* write header */
write_header(startrow,startcol,endcol,header[0],REV_VID);

/* write data labels */
for(row=startrow+5,i=0; i<8; i++,row++)
  write_string(row,startcol+3,label[i],REV_VID);

/* write data */
/* write wind resistance */
sprintf(string,"%8.0f",res_dat[0]);
write_string(startrow+5,startcol+2+strlen(label[4])+2,string,REV_VID);

/* write wave height */
sprintf(string,"%8.1f",res_dat[1]);
write_string(startrow+6,startcol+2+strlen(label[4])+2,string,REV_VID);

/* write added resistance */
sprintf(string,"%8.0f",res_dat[2]);
write_string(startrow+7,startcol+2+strlen(label[4])+2,string,REV_VID);

/* write hull resistance */
sprintf(string,"%8.0f",res_dat[3]);
write_string(startrow+8,startcol+2+strlen(label[4])+2,string,REV_VID);
```

```
  /* write prop resistance */
  sprintf(string,"%8.0f",res_dat[4]);
  write_string(startrow+9,startcol+2+strlen(label[4])+2,string,REV_VID);

  /* write total tow resistance */
  sprintf(string,"%8.0f",tow_res);
  write_string(startrow+10,startcol+2+strlen(label[4])+2,string,REV_VID);

  /* write hawser resistance */
  sprintf(string,"%8.0f",haw_res);
  write_string(startrow+11,startcol+2+strlen(label[4])+2,string,REV_VID);

  /* write mean tension */
  sprintf(string,"%8.0f",*tension);
  write_string(startrow+12,startcol+2+strlen(label[4])+2,string,REV_VID);

  /* write units */
  for(row=startrow+5,i=0; i<8; i++,row++)
    write_string(row,endcol-5,"lbs",REV_VID);
  write_string(startrow+6,endcol-5,"ft ",REV_VID);

  /* write footer */
  start = (endcol-startcol-strlen(footer))/2 + startcol;
  write_string(endrow,start,footer,REV_VID);

  pause(startrow,startcol,endrow,endcol,footer,REV_VID);
  return ;
}

/*****************************************************************************
     This function computes the added resistance due to waves. Data from
 Figure G-2 of the USN Towing Manual are read in from the file "RS_WVHT.DAT"
 and are linearly interpolated for the specified wave height. The first
 argument is the desired wave height, the second is the curve number as
 determined from Table G-2; the third argument is the result, the added
 resistance.
*****************************************************************************/
added_res(wave_ht,curve,resistance)
float wave_ht;                          /* Wave height, feet              */
float *resistance;                      /* Added resistance, lbs          */
int curve;                              /* Curve no. for added resistance */
{
   FILE *in;
   char string[81],dummy[81],text1[36],text2[36];
   char ch;
   int i;                               /* Dummy counter                  */
   int status;                          /* Return value for "stofa"       */
   int n;                               /* Required arg for "stofa"       */
   int index;                           /* Index for table lookup         */
   float test;                          /* Test value for if statement    */
   float lower_dat[4], upper_dat[4];    /* Bracketing data points         */
   float curve_dat[4];                  /* Exact or interpolated data     */

   in = fopen("rs_wvht.dat","r");       /* Open file */
```

```
   index = wave_ht/1;              /* Compute test value for if statement */
   test = wave_ht - (float)index; /*                 "                     */

   /* Read and discard text lines of file  */
   fgets(dummy,81,in);
   fgets(dummy,81,in);
   fgets(dummy,81,in);
   fgets(dummy,81,in);

   if (wave_ht == 0.0) {
      fgets(string,81,in);
      status = stofa(string,curve_dat,&n,4);
   }

   else if (test > 0.0) {  /* Need interpolation */
      for (i=0; i<index; i++)
         fgets(dummy,81,in);  /* Read & discard data up to the desired line */
      fgets(string,81,in);    /* Read line of file */
      status = stofa(string,lower_dat,&n,4);
      fgets(string,81,in);    /* Read next line of file */
      status = stofa(string,upper_dat,&n,4);
      for(i=0; i<4; i++) {
         curve_dat[i] =lower_dat[i] + (wave_ht-index) * (upper_dat[i]
                       - lower_dat[i]);
      }
   }
   else if (test == 0.0) {   /* No interpolation needed */
      for (i=0; i<index; i++)
         fgets(dummy,81,in);  /* Read & discard data up to the desired line */
      fgets(string,81,in);    /* Read line of file */
      status = stofa(string,curve_dat,&n,4);
   }


   /* Check consistency of data: */
   if ( curve_dat[curve] > 1.0e30)
   {
     sprintf(text1,"Wave height outside data range");
     sprintf(text2,"for this curve.");
     display_error(ERROR,text1,text2);
     while( !(ch=getchar()) ) ;
     fclose(in);
     return -1;
   }

   *resistance = curve_dat[curve];     /* Passes the appropriate value  */
                                       /* back to the calling function. */
   fclose(in);
   return 0;
}

/*****************************************************************************
     This function finds the hull resistance of the ship (RH). Data
```

```
 from Figure G-1 of the USN Towing Manual are read in from the file
 "RH_DISP.DAT" and are linearly interpolated for the specified towing speed.
 The first argument is the desired towing speed, the second is the curve
 number as determined from Table G-2, the third argument is the displacement;
 the fourth argument is the result, the hull resistance.
*****************************************************************************/
hull_resist(speed,curve,disp,resistance)
float speed;                          /* Towing speed, kts               */
int curve;                            /* Curve no. for added resistance  */
float disp;                           /* Displacement, tons              */
float *resistance;                    /* Hull resistance, lbs            */
{
   FILE *in;
   char string[81],dummy[81],text1[36],text2[36];
   char ch;
   int i;                             /* Dummy counter                   */
   int status;                        /* Return value for "stofa"        */
   int n;                             /* Required arg for "stofa"        */
   int index;                         /* Index for table lookup          */
   float test;                        /* Test value for if statement     */
   float lower_dat[6], upper_dat[6];  /* Bracketing data points          */
   float curve_dat[6];                /* Exact or interpolated data      */

   in = fopen("rh_disp.dat","r");     /* Open file */

   index = speed/1;                /* Compute test value for if statement */
   test = speed - (float)index;    /*                 "                   */

   /* Read and discard text lines of file  */
   fgets(dummy,81,in);
   fgets(dummy,81,in);
   fgets(dummy,81,in);

   if (speed == 0.0) {
      fgets(string,81,in);
      status = stofa(string,curve_dat,&n,6);
   }

   else if (test > 0.0) {  /* Need interpolation */
      for (i=0; i<index; i++)
         fgets(dummy,81,in);  /* Read & discard data up to the desired line */
      fgets(string,81,in);    /* Read line of file */
      status = stofa(string,lower_dat,&n,6);
      fgets(string,81,in);    /* Read next line of file */
      status = stofa(string,upper_dat,&n,6);
      for(i=0; i<6; i++) {
         curve_dat[i] =lower_dat[i] + (speed-index) * (upper_dat[i]
                       - lower_dat[i]);
      }
   }
   else if (test == 0.0) {   /* No interpolation needed */
      for (i=0; i<index; i++)
         fgets(dummy,81,in);  /* Read & discard data up to the desired line */
      fgets(string,81,in);    /* Read line of file */
```

```
      status = stofa(string,curve_dat,&n,6);
   }

   *resistance = 1.25*curve_dat[curve]*disp;   /* incl 25% for hull fouling */

   fclose(in);
   return 0;
}

/*****************************************************************************
     This function computes the significant wave height as a function of the
 maximum expected wind speed, using the Pierson-Moskowitz sea spectrum.
*****************************************************************************/
wave_height(wind_spd,height)
float wind_spd;          /* wind speed, kts */
float *height;           /* wave height, ft */
{
  float alpha=8.1e-3;    /* P-M nondimensional parameter */
  float beta=0.74;       /* P-M nondimensional parameter */
  float g=32.17;         /* gravitational constant */
  float coef;

  coef = 2.0/g*sqrt(alpha/beta);

  *height = coef*(wind_spd*wind_spd)*(1.689*1.689);      /* wave height in ft */
}

/*****************************************************************************
     This function finds the hawser resistance.
*****************************************************************************/
void hawser_resist(tug_data,speed,tow_res,wire_resist)
float *tug_data;         /* tug data array          */
float speed;             /* tow speed, kts          */
float tow_res;           /* tow resistance, lbs     */
float *wire_resist;      /* hawser resistance, lbs */
{
  int i;                        /* dummy counter                        */
  static float rho=1.9905;      /* density of sea water, slugs/ft^3  */
  static float coef_n=1.4;      /* normal drag coefficient              */
  static float coef_t=0.015;    /* tangential drag coefficient          */
  float T;                      /* static hawser tension                */
  float p;                      /* weight of hawser in water, lbs/ft */
  float d;                      /* hawser diameter, ft                  */
  float phi;                    /* cable inclination angle              */
  float vtow;                   /* tow speed, ft/sec                    */
  float sinphi;                 /* sin(phi)                             */
  float cosphi;                 /* cos(phi)                             */
  float k1,drag,olddrag;        /* intermediate results                 */

  if (tug_data[1]==2.0)
    p=6.4293;
  else if (tug_data[1]==2.25)
    p=8.1432;
  d=tug_data[1]/12.0;
```

```
  vtow=speed*1.689;
  T=tow_res;

  /* compute hawser resistance iteratively until
     equilibrium is reached  */
  for (i=0; i<20; i++)
  (
    k1 = rho*d*vtow*vtow*T/p;
    phi=atan(p*tug_data[2]/2.0/T);
    sinphi=sin(phi);
    cosphi=cos(phi);

    drag=k1*(coef_n*(pow(sinphi,4)/cosphi+cosphi*(sinphi*sinphi+2.0)
                -2.0)+PI*coef_t*sinphi);
    T = tow_res + drag;
    if (i==0)
    (
      olddrag=drag;
      continue;
    )
    if ( fabs(drag-olddrag) < 0.1) break;
    else olddrag=drag;
  )
  *wire_resist=drag;
}

/*********************************************************************
     This function finds the ship resistance.
*********************************************************************/
void ship_resist(tow_data,ship_data,res_dat,tow_res)
float *tow_data;
float *ship_data;
float *res_dat;
float *tow_res;
(
  float k;               /* heading coefficient           */
  float rel_wind_spd2;   /* relative wind speed squared   */
  float wind_res;        /* wind resistance, lbs          */
  float vtow2;           /* tow speed squared             */
  float vwind2;          /* wind speed squared            */
  float alpha;           /* supp. angle of rel wind direction */
  float wave_ht;         /* wave height, ft               */
  float add_res;         /* added resistance due to waves */
  float hull_res;        /* calm water hull resistance    */
  float prop_res;        /* propeller resistance          */
  float prop_area;       /* propeller area                */
  int curve,status;

  /* determine heading coefficient */
  if (tow_data[3]<20.0)
    k = 1.0;
  else if (tow_data[3]>=20.0 && tow_data[3]<40.0)
    k = 1.2;
  else if (tow_data[3]>=40.0 && tow_data[3]<=90.0)
```

```
  k = 0.4;
else
  k = 0.0;

vtow2 = square(tow_data[1]);

/***************************/
/* compute wind resistance */
/***************************/

if (tow_data[3]==0.0)
  rel_wind_spd2 = square(tow_data[1]+tow_data[2]);
else
{
  vwind2 = square(tow_data[2]);
  alpha = (180.0 - tow_data[3])*PI/180.0;
  rel_wind_spd2 = vwind2+vtow2-2.0*tow_data[1]*tow_data[2]*cos(alpha);
}

wind_res=0.00506*ship_data[2]*ship_data[1]*k*rel_wind_spd2;
res_dat[0]=wind_res;

/****************************************/
/* compute added resistance due to waves */
/****************************************/

wave_height(tow_data[2],&wave_ht);
res_dat[1]=wave_ht;

curve = (int) ship_data[5];
status = added_res(wave_ht,curve,&add_res);
if (status<0)
  add_res=0.0;
res_dat[2]=add_res;

/*************************************/
/* compute calm water hull resistance */
/*************************************/

curve = (int) ship_data[4];
status = hull_resist(tow_data[1],curve,tow_data[0],&hull_res);
res_dat[3]=hull_res;

/*********************************/
/* compute propeller resistance */
/*********************************/

prop_area=ship_data[3];
if (tow_data[4]==LOCKED) ;
else if (tow_data[4]==TRAILING)
  prop_area *= 0.5;
else if (tow_data[4]==REMOVED)
  prop_area = 0.0;
```

```
  prop_res = 3.737*prop_area*vtow2;
  res_dat[4]=prop_res;

  /*********************************/
  /* compute total tow resistance */
  /*********************************/

  *tow_res = wind_res + add_res + hull_res + prop_res;
}
```

```
/*****************************************************************************
  File: ext.c
  Author: Todd J. Peltzer
  Last update: 3 May 1989

    This file contains the functions which compute the extreme tension
  for a given combination of tug, tow, wind, tow speed, hawser scope,
  and heading angle.
-----------------------------------------------------------------------------
  Functions:
    extreme()
    set_towspd()
    set_wind()
    set_hdg()
    set_scope()
    set_coef()
    curvindex()
    stoia()
    get_tension()
    get_ext_opt()
    comp_ten_curv()
    show_std_curve()
    show_speed()
    show_length()
    clip()
    wait_plt()
*****************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "stdlib.h"
#include "keydef.h"   /* Define aux byte values for IBM keyboard */
#include "video.h"
#include "math.h"

typedef struct Subarray
{
  short c[4][6];
} SUBARRAY;

typedef struct Table
{
  SUBARRAY s[4][3];
} TABLE;

typedef struct
{
  int startcol;
  int endcol;
} data_box;

static data_box input[] =
{
        {50,57}         /* tension */
};
```

```
static char *label2[] =
{
  "Select basis for computing",
  "extreme tension:",
  "Actual mean tension (lbs): "
};

static char *label1[] =
{
  "DYNAMIC TENSION",
  "Tug:",
  "Tow:",
  "Tabulated tow:",
  "                  Actual  Tabulated",
  "Displacement:",
  "Tow speed:",
  "Wind speed:",
  "Wind direction:",
  "Hawser scope:",
  "Curve number:",
  "Mean tension:",
  "Dynamic tension:",
  "Extreme tension:"
};

static char *menu[] =
{
  "1) Estimated mean tension",
  "2) Actual mean tension   "
};

static char *menu1[] =
{
  "1) Show standard tension curve  ",
  "2) Show effects of tow speed    ",
  "3) Show effects of hawser length",
  "4) Return to PROGRAM OPTIONS    "
};

static char *menu2[] =
{
  "1) Show graph       ",
  "2) Return to OPTIONS"
};

static char *footer[] =
{
  " Please wait ",
  " Press INS to continue "
};

static char *tabtow[] =
{
```

```
  "YRBM",
  "FFG 1",
  "DD 963",
  "AE 26",
  "LHA 1",
  "CVN 65"
};

static char ext_hdr[] = { "OPTIONS" };

static int startrow=2;
static int startcol=20;
static int endrow=22;
static int endcol=60;
float fa[4][25],fb[4][25],fc[4][25],fd[4][25];

static float towdisp[] = { 650.0, 3200.0, 6700.0,
                           20000.0, 40000.0, 85000.0 };
static float towspd[] = { 3.0, 6.0, 9.0 };
static float windspd[] = { 15.0, 20.0, 25.0, 30.0, };
static float winddir[] = { 0, 60, 120, 180 };
static float scope[] = { 1000, 1200, 1500, 1800, 2100 };

void extreme(),curvindex(),set_towspd(),set_wind();
void set_hdg(),set_scope(),get_tension(),get_ext_opt();
void show_std_curve(),show_speed(),show_length();
void comp_ten_curv(),set_coef(),clip(),set_tugtype();
void set_towtype(),wait_plt();

extern char tug[15];           /* tug class                      */
extern char hull_no[24];       /* hull number entered by user    */
extern float tug_data[5];      /* array to store tug data        */
extern float tow_data[5];      /* array to store tow data        */
extern float ship_data[6];     /* array to store Table G-2 data  */
extern float dock_data[7];     /* array to store drydock data    */
extern float barge_data[8];    /* array to store barge data      */
extern float ext_data[10];     /* extreme tension results        */
extern float spd_data[3][4];   /* tow speed effects results      */
extern float lgth_data[3][4];  /* hawser length effects results */

TABLE t[18];

/*************************************************************************
  This function computes the extreme tension based on the given
  parameters.
*************************************************************************/
void extreme(type,tension,extr_ten,curve,flag)
int type;
float *tension;
float *extr_ten;
int *curve;
int *flag;
{
  char inline[81];
```

```
char text1[39],text2[39];
char string[26];
float n,x;
float tow_res,haw_res,res_dat[5];
int i,j;
int k=0;
int m=0;
int nn=0;
int n1;
int nbr,status,choice;
short array[15];
FILE *in;
int tugtype,towtype,windspeed,towspeed,head,length,tugtow;
int row, col, start, start1, key;

in=fopen("curve.tab","r");

/* draw background and border */
draw_window(startrow,startcol,endrow,endcol,DOUBLE,REV_VID);

/* write header */
write_header(startrow,startcol,endcol,label1[0],REV_VID);

/* get choice for basis */
for (i=0; i<2; i++)
(
  start = (endcol-startcol-strlen(label2[i]))/2 + startcol;
  write_string(startrow+8+i,start,label2[i],REV_VID);
)
start = (endcol-startcol-strlen(menu[0]))/2 + startcol;
choice = popup(menu,"12",2,startrow+10,start-2,NONE,REV_VID,0);
ext_data[9] = (float)choice;

clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
if (choice==0)    /* use estimated tension from App. G method */
(
  if (*flag==1)
  (
    if (type==SHIP)
    {
      ship_resist(tow_data,ship_data,res_dat,&tow_res);
      hawser_resist(tug_data,tow_data[1],tow_res,&haw_res);
      *tension = tow_res + haw_res;
    }
    else if (type==DOCK)
      dock_resist(tug_data,dock_data,tow_data,tension);

    else          /* type==BARGE */
      barge_resist(tug_data,barge_data,tow_data,tension);
  )
  *flag=0;
)
else
(
```

```
  *flag=1;
  /* prompt for actual mean tension */
  write_string(startrow+8,startcol+3,label2[2],REV_VID);
  vid_box(startrow+8,startcol+3+strlen(label2[2]),7);
  get_tension(startrow+8,startcol+3+strlen(label2[2]),tension);
  clear(startrow+8,startcol+1,endrow-3,endcol-1,REV_VID);
}

/* write labels */
for (i=0; i<13; i++)
  write_string(startrow+5+i,startcol+3,label1[i+1],REV_VID);

/* write footer */
start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
write_string(endrow,start,footer[0],BLINK_REV_VID);

/* set tugtype */
set_tugtype(&tugtype);

/* determine towtype */
set_towtype(&towtype);
ext_data[0]=(float)towtype;
ext_data[1]=towdisp[towtype];

/* set tow speed */
set_towspd(tow_data[1],&towspeed);
ext_data[2]=towspd[towspeed];

/* set wind speed */
set_wind(tow_data[2],&windspeed);
ext_data[3]=windspd[windspeed];

/* set heading angle */
set_hdg(tow_data[3],&head);
ext_data[4]=winddir[head];

/* set hawser scope */
set_scope(tug_data[2],&length);
ext_data[5]=scope[length];

/* compute tug-tow combination index */
tugtow = towtype + 6*tugtype;

/* set up coefficient array */
set_coef();

/* read in data table for curve numbers from curve.tab */
for (nn=0; nn<18; nn++)
(
  for (m=0; m<4; m++)
  {
    for (k=0; k<4; k++)
    (
      fgets(inline,81,in);
```

```
      while(inline[0]== '!')
        fgets(inline,81,in);       /* skip comments, headers, etc. */

      /* convert string to array of short integers */
      status=stoia(inline,array,&nbr,15);

      /* assign curve numbers to data structure */
      for(j=0; j<3; j++)
      {
        for (i=0; i<5; i++)
          t[nn].s[m][j].c[k][i] = array[i + j*5];
      }
    }
  }
}
/* compute i,j corresponding to curve number */
*curve = t[tugtow].s[windspeed][towspeed].c[head][length];
ext_data[6] = *curve;
ext_data[7] = *tension/1000.0;

curvindex(*curve,&i,&j);

x=(*tension)/100000.0;

/***************************/
/* compute extreme tension */
/***************************/
*extr_ten = 100.0*(x+fa[i][j]*x/(1.0+fb[i][j]*x)
         + fc[i][j]*x*x*x*x/(1.0+fd[i][j]*x*x*x*x));

ext_data[8] = *extr_ten;

/* check magnitude of extreme tension */
if (*extr_ten>400.0)
{
  /* erase blinking footer */
  start = (endcol-startcol-strlen(footer[0]))/2 + startcol;
  for (k=0; k< strlen(footer[0]); k++)
    write_char(endrow,start,205,REV_VID);

  /* display error message; return to previous menu */
  sprintf(text1,"Predicted extreme tension > 400 kips");
  sprintf(text2,"==> SLOW DOWN, PAYOUT, OR BOTH <==");
  display_error(ERROR,text1,text2);
  cursor_off();
  return ;
}

/* print result */
write_string(startrow+5,startcol+4+strlen(label1[1]),tug,REV_VID);
write_string(startrow+6,startcol+4+strlen(label1[2]),hull_no,REV_VID);
write_string(startrow+7,startcol+4+strlen(label1[3]),tabtow[towtype],REV_VID);

sprintf(string,"%8.0f  %8.0f",tow_data[0],towdisp[towtype]);
```

```
  write_string(startrow+9,startcol+3+15,string,REV_VID);

  sprintf(string,"%8.1f  %8.1f",tow_data[1],towspd[towspeed]);
  write_string(startrow+10,startcol+3+15,string,REV_VID);

  sprintf(string,"%8.0f  %8.0f",tow_data[2],windspd[windspeed]);
  write_string(startrow+11,startcol+3+15,string,REV_VID);

  sprintf(string,"%8.0f  %8.0f",tow_data[3],winddir[head]);
  write_string(startrow+12,startcol+3+15,string,REV_VID);

  sprintf(string,"%8.0f  %8.0f",tug_data[2],scope[length]);
  write_string(startrow+13,startcol+3+15,string,REV_VID);

  sprintf(string,"%d",*curve);
  write_string(startrow+14,startcol+15+strlen(label1[10]),string,REV_VID);

  sprintf(string,"%10.2f",*tension/1000.0);
  write_string(startrow+15,startcol+4+3+strlen(label1[11]),string,REV_VID);

  x = *extr_ten - *tension/1000.0;
  if (x < 0.0 )
    x = 0.0;
  sprintf(string,"%10.2f",x);
  write_string(startrow+16,startcol+4+strlen(label1[12]),string,REV_VID);

  sprintf(string,"%10.2f",*extr_ten);
  write_string(startrow+17,startcol+4+strlen(label1[13]),string,REV_VID);

  for (i=0; i<3; i++)
    write_string(startrow+15+i,startcol+15+strlen(label1[13]),"kips",REV_VID);

  pause(startrow,startcol,endrow,endcol,footer[1],REV_VID);
  return;
}

/*******************************************
  This function sets the tug type index.
*******************************************/
void set_tugtype(tugtype)
int *tugtype;
{
  if (tug_data[0]==0)
    *tugtype=0;
  else if (tug_data[0]==1 || tug_data[0]==2)
    *tugtype=1;
  else
    *tugtype=2;
}

/*******************************************
  This function sets the tow type index.
*******************************************/
void set_towtype(towtype)
```

```
int *towtype;
{
  int j=0;

  while (1)
  {
    if (tow_data[0] < towdisp[0]) {
      *towtype=0;
      break;
    }
    else if (tow_data[0]==towdisp[j]) {
      *towtype=j;
      break;
    }
    else if (tow_data[0] > towdisp[j]) {
      if (j<5) {
        if (tow_data[0] > towdisp[j] && tow_data[0] < towdisp[j+1]) {
          if (tow_data[0] > 0.75*towdisp[j+1] ) {
            *towtype=j+1;
            break;
          }
          else {
            *towtype=j;
            break;
          }
        }
        else {
          j++;
          continue;
        }
      }
      else {
        *towtype=j;
        break;
      }
    }
    else {
      j++;
      continue;
    }
  }
}

/*****************************************
  This function sets the tow speed index.
*****************************************/
void set_towspd(speed,towspeed)
float speed;              /* tow speed */
int *towspeed;            /* tow speed index */
{
  if (speed<4.5)
    *towspeed=0;
  else if (speed>=4.5 && speed<7.5)
    *towspeed=1;
```

```
  else
    *towspeed=2;
}

/*****************************************
  This function sets the wind speed index.
*****************************************/
void set_wind(wind,windspeed)
float wind;                /* wind speed */
int *windspeed;            /* wind speed index */
{
  if (tow_data[2]<17.5)
    *windspeed=0;
  else if (wind>=17.5 && wind<22.5)
    *windspeed=1;
  else if (wind>=22.5 && wind<27.5)
    *windspeed=2;
  else
    *windspeed=3;
}

/*****************************************
  This function sets the heading index.
*****************************************/
void set_hdg(hdg,head)
float hdg;                 /* heading angle */
int *head;                 /* heading angle index */
{
  if (tow_data[3]<40.0)
    *head=0;
  else if (hdg>=40.0 && hdg<70.0)
    *head=1;
  else if (hdg>=110.0 && hdg<140.0)
    *head=2;
  else
    *head=3;
}

/*****************************************
  This function sets the length index.
*****************************************/
void set_scope(scope,length)
float scope;               /* hawser length */
int *length;               /* hawser length index */
{
  char text1[39],text2[39];

  if (scope<1000.0)
  {
    /* no data for shorter scope */
    sprintf(text1,"No data for scopes less than 1000 ft");
    display_error(WARN,text1," ");
  }
  else if (scope>=1000.0 && scope<1200.0)
```

```
    *length=0;
  else if (scope>=1200.0 && scope<1500.0)
    *length=1;
  else if (scope>=1500.0 && scope<1800.0)
    *length=2;
  else if (scope>=1800.0 && scope<2100.0)
    *length=3;
  else
    *length=4;
)

/*************************************************************************
  This function sets up the coefficient arrays for computing the
  standard curves for extreme tension.
*************************************************************************/
void set_coef()
{
  int i;
  float n,n1;

  for (i=0; i<25; i++)
  {
    n= (float)i;
    fa[0][i] = 0.3*n;
    if (i>7)
    {
      n1 = n-7.0;
      fa[0][i] += 0.00013*n1*n1*n1*n1;
    )
    fb[0][i] = 0.0;
    fc[0][i] = 0.0;
    fd[0][i] = 0.0;
    fa[1][i] = 0.6*n+0.0003*n*n*n;
    fb[1][i] = 3.0;
    fc[1][i] = 0.0;
    fd[1][i] = 0.0;
    fa[2][i] = 0.11*n;
    if (i>7)
    {
      n1 = n-7.0;
      fa[2][i] += 0.0002*n1*n1*n1*n1;
    }
    fb[2][i] = 3.0+0.11*n;
    fc[2][i] = 20.0*fa[2][i];
    fd[2][i] = 5.0;
    fa[3][i] = 0.03*n;
    if (i>7)
    {
      n1 = n-7.0;
      fa[3][i] += 0.00014*n1*n1*n1*n1;
    )
    fb[3][i] = 5.0+0.04*n;
    fc[3][i] = 80.0*fa[3][i];
    fd[3][i] = 6.0;
```

```
  )
)

/*************************************************************************
  This function takes a "standard curve" number and converts it to
  its component indices.
*************************************************************************/
void curvindex(curve,i,j)
int curve;
int *i;
int *j;
{
  if (curve<25) *i=0;
  else if (curve>24 && curve<50) *i=1;
  else if (curve>49 && curve<75) *i=2;
  else if (curve>74 && curve<100) *i=3;
  (*j) = curve - 25*(*i);
}

/*************************************************************************
     This function gets the user's input for the data requested in the
  function extreme(). Uses screen_getstring() for screen I/O. Some data
  checking is performed for each data type.
*************************************************************************/
void get_tension(row,col,data)
int row;
int col;
float *data;
{
  int nbr;                    /* required arg for stofa()       */
  int status;                 /* takes return value for stofa() */
  char string[24];            /* input string                   */
  char text1[39], text2[39]; /* error message text             */
  int j, k;                   /* counters                       */

  cursor_on();                /* turn on cursor                 */
  goto_xy(row,col);           /* move cursor to start of box    */

  /* get user input */
  screen_getstrg(0,row,col,string,NORM_VID,input);

  /* check for valid numeric input */
  for (k=0; k<strlen(string); k++)
  {
    if ( string[k]=='.' ) ;
    else if (string[k]<'0' || string[k]>'9')
    {
      sprintf(text1,"Invalid entry; try again");
      display_error(ERROR,text1," ");
      vid_box(row,col,7);
      get_tension(row,col,data);
      return ;
    }
  }
```

```
  /* if input string contains valid numbers only, */
  /*   convert string to float */
  status = stofa(string,data,&nbr,1);

  /************************************************/
  /*            check data for consistency        */
  /************************************************/
  if (*data<0.0)
  {
    sprintf(text1,"Invalid entry; try again");
    display_error(ERROR,text1," ");
    vid_box(row,col,7);
    get_tension(row,col,data);
    return ;
  }
  if (*data>100000)
  {
    sprintf(text1,"Mean tension is beyond the valid");
    sprintf(text2,"range of input--please try again");
    display_error(ERROR,text1,text2);
    vid_box(row,col,7);
    get_tension(row,col,data);
    return;
  }

  cursor_off();
  goto_xy(0,0);
}

/*****************************************************************************
     This function displays a popup menu which lists the graphics options
  available in this module.
*****************************************************************************/
void get_ext_opt(flag,type,curve,mean,ext,hilite)
int flag;               /* indicates estimated or actual mean selected */
int type,curve;         /* tow type, std curve number */
float mean,ext;         /* mean, extreme tensions */
int hilite;             /* popup menu choice to highlight */
{
  int choice,start;
  int i,j;
  char text1[39],text2[39];

  cursor_off();

  /* draw background and border */
  draw_window(startrow,startcol,endrow,endcol,DOUBLE,REV_VID);

  /* display popup menu */
  start = (endcol-startcol-strlen(menu1[0]))/2 + startcol;
  write_header(startrow,startcol,endcol,ext_hdr,REV_VID);
  choice=popup(menu1,"1234",4,startrow+5,start-2,NONE,REV_VID,hilite);

  if (choice==0)
```

```
  {
    curvindex(curve,&i,&j);
    show_std_curve(i,j,curve,mean,ext);
    get_ext_opt(flag,type,curve,mean,ext,1);
  }
  else if (choice==1)
  {
    curvindex(curve,&i,&j);
    if (flag)
    {
      sprintf(text1,"This option uses only predicted");
      sprintf(text2,"mean tensions");
      display_error(WARN,text1,text2);
      cursor_off();
    }
    show_speed(type,i,j,curve);
    get_ext_opt(flag,type,curve,mean,ext,2);
  }
  else if (choice==2)
  {
    curvindex(curve,&i,&j);
    if (flag)
    {
      sprintf(text1,"This option uses only predicted");
      sprintf(text2,"mean tensions");
      display_error(WARN,text1,text2);
      cursor_off();
    }
    show_length(type,i,j,curve);
    get_ext_opt(flag,type,curve,mean,ext,3);
  }
  else return;
}

/*************************************************************************
     This function displays the extreme tension curve corresponding to
  the conditions specified for the given tug and tow.
*************************************************************************/
void show_std_curve(i,j,curve,mean,ext)
int i,j,curve;
float mean,ext;
{
  int k;                        /* dummy counter */
  float x[121],y[121];          /* arrays to hold x, y coordinates */
  float x1,y1,x2,y2;            /* coordinates for drawing lines */
  char labx[25],laby[25];       /* strings for x, y axis labels */
  short kd=2;                   /* graph "kode" */
  short npts;                   /* number of points to plot */
  short ntx=2,nty=1;            /* skip factor for axis numbers */
  short itype;                  /* designates x or y axis label in label() */
  short nsym;                   /* symbol code */
  float x0i=1.0764;
  float y0i=1.0;
  float xli=6.624;
```

```
float yli=4.35;
float xmn=0.0;              /* min x value */
float xmx=120.0;            /* max x value */
float xtc=10.0;             /* x axis tic value */
float ymn=0.0;              /* min y value */
float ymx=400.0;            /* max y value */
float ytc=50.0;             /* y axis tic value */
float xxmax;                /* x corresponding to y=400.0 kips */
float xsm,ysm;              /* smallest x,y in world coordinates */

xsm = xmn - (x0i/xli)*(xmx-xmn);
ysm = ymn - (y0i/yli)*(ymx-ymn);
mean /= 1000.0;                         /* convert to kips */
setplt();                               /* initialize graphics */
locate(&x0i,&y0i,&xli,&yli);            /* set position on screen */
xyaxis(&xmn,&xmx,&xtc,&ymn,&ymx,
      &ytc,&xmn,&ymn,&kd);              /* draw axes */
axnum(&ntx,&nty);                       /* write axis numbers */
itype=1;
sprintf(labx,"Mean Tension (kips)_");
label(&itype,labx);                     /* write x-axis label */
itype=2;
sprintf(laby,"Extreme Tension (kips)_");
label(&itype,laby);                     /* write y-axis label */
comp_ten_curv(i,j,x,y);                 /* compute curve pts  */
clip(x,y);                              /* clip curve above 400kips */

npts=122;
line3(&npts,x,y);                       /* draw line thru pts */

/* don't plot points if mean tension > 120 kips */
if (mean<=120.0)
{
  npts=1;                               /* plot symbol at computed  */
  x[0]= mean;                           /* value of extreme tension */
  y[0]= ext;
  nsym=3;                               /* use diamond as data symbol */
  symplt(&npts,x,y,&nsym);

  x1= mean;                             /* draw line */
  y1=0.0;
  x2= mean;
  y2= ext;
  drawu(&x1,&y1,&x2,&y2);

  x1=0.0;                               /* draw line */
  y1= ext;
  x2= mean;
  y2= ext;
  drawu(&x1,&y1,&x2,&y2);
)

x1=5.0;                                 /* write label */
y1= 380.0;
```

```
  sprintf(labx,"Curve No. %d_",curve);
  labely(&x1,&y1,labx);

  /* wait for keystroke, then exit */
  wait_plt();
}

/*************************************************************************
     This function shows the effects of speed on extreme tension by
  computing the mean tension for Vtow +/- 1.0 kts, finding the appropriate
  extreme tension curves, and displaying a graph of all three curves.
  A summary of results is displayed prior to displaying the graphs.
*************************************************************************/
void show_speed(type,i,j,curve)
int type;
int i,j,curve;
{
  float x_hi[121],x_lo[121],y_hi[121],y_lo[121];
  float x[121],y[121];
  float tension_hi,tension_lo,extr_hi,extr_lo;
  float mean,ext;
  float spdhi,spdlo,oldspd;
  int tugtype,towtype,tugtow,windspeed,head,length;
  int towspdhi,towspdlo;
  int curve_hi,i_hi,j_hi;
  int curve_lo,i_lo,j_lo;
  int k;                        /* dummy counter */
  float x1,y1,x2,y2;            /* coordinates for drawing */
  char labx[25],laby[25];       /* strings for x, y axis labels */
  short kd=2;                   /* graph "kode" */
  short npts;                   /* number of points to plot */
  short ntx=2,nty=1;            /* skip factor for axis numbers */
  short itype;                  /* designates x or y axis label in label() */
  short nsym;                   /* symbol code */
  float x0i=1.0764;
  float y0i=1.0;
  float xli=6.624;
  float yli=4.35;
  float xmn=0.0;                /* min x value */
  float xmx=120.0;              /* max x value */
  float xtc=10.0;               /* x axis tic value */
  float ymn=0.0;                /* min y value */
  float ymx=400.0;              /* max y value */
  float ytc=50.0;               /* y axis tic value */
  float xxmax;                  /* x corresponding to y=400.0 kips */
  float res_dat[5];
  float tow_res,haw_res;
  float xx;
  float xsm,ysm;                /* smallest x,y in world coordinates */
  int start,choice;

  xsm = xmn - (x0i/xli)*(xmx-xmn);
  ysm = ymn - (y0i/yli)*(ymx-ymn);
```

```
/* write message */
draw_window(startrow+11,startcol+12,startrow+13,startcol+27,
              SINGLE,REV_VID);
sprintf(labx,"Please wait");
start = (endcol-startcol-strlen(labx))/2 + startcol;
write_string(startrow+12,start,labx,BLINK_REV_VID);

/* set speeds */
oldspd=tow_data[1];
spdlo = tow_data[1]-1.0;
if (spdlo <= 0.0)
  spdlo = 1.0;
spdhi = tow_data[1]+1.0;
if (spdhi > 12.0)
  spdhi = 12.0;

/* compute mean tension */
if (type==SHIP)
{
  /* -- for higer speed */
  tow_data[1]=spdhi;
  ship_resist(tow_data,ship_data,res_dat,&tow_res);
  hawser_resist(tug_data,spdhi,tow_res,&haw_res);
  tension_hi = tow_res + haw_res;

  /* -- for lower speed */
  tow_data[1]=spdlo;
  ship_resist(tow_data,ship_data,res_dat,&tow_res);
  hawser_resist(tug_data,spdlo,tow_res,&haw_res);
  tension_lo = tow_res + haw_res;

  /* -- for given speed */
  tow_data[1]=oldspd;
  ship_resist(tow_data,ship_data,res_dat,&tow_res);
  hawser_resist(tug_data,oldspd,tow_res,&haw_res);
  mean = tow_res + haw_res;
}
else if (type==DOCK)
{
  /* -- for higer speed */
  tow_data[1]=spdhi;
  dock_resist(tug_data,dock_data,tow_data,&tension_hi);

  /* -- for lower speed */
  tow_data[1]=spdlo;
  dock_resist(tug_data,dock_data,tow_data,&tension_lo);

  /* -- for given speed */
  tow_data[1]=oldspd;
  dock_resist(tug_data,dock_data,tow_data,&mean);
}
else          /* type==BARGE */
{
  /* -- for higer speed */
```

```
  tow_data[1]=spdhi;
  barge_resist(tug_data,barge_data,tow_data,&tension_hi);

  /* -- for lower speed */
  tow_data[1]=spdlo;
  barge_resist(tug_data,barge_data,tow_data,&tension_lo);

  /* -- for given speed */
  tow_data[1]=oldspd;
  barge_resist(tug_data,barge_data,tow_data,&mean);
}

/* convert to kips */
tension_lo /= 1000.0;
tension_hi /= 1000.0;
mean /= 1000.0;

/* set common indices */
set_tugtype(&tugtype);
set_towtype(&towtype);
tugtow = towtype + 6*tugtype;

set_wind(tow_data[2],&windspeed);
set_hdg(tow_data[3],&head);
set_scope(tug_data[2],&length);

/* find curve for spdhi */
set_towspd(spdhi,&towspdhi);
curve_hi=t[tugtow].s[windspeed][towspdhi].c[head][length];
curvindex(curve_hi,&i_hi,&j_hi);

/* find curve for spdlo */
set_towspd(spdlo,&towspdlo);
curve_lo=t[tugtow].s[windspeed][towspdlo].c[head][length];
curvindex(curve_lo,&i_lo,&j_lo);

/* compute tension arrays */
comp_ten_curv(i_hi,j_hi,x_hi,y_hi);
clip(x_hi,y_hi);
comp_ten_curv(i_lo,j_lo,x_lo,y_lo);
clip(x_lo,y_lo);
comp_ten_curv(i,j,x,y);
clip(x,y);

/* compute extremes */
xx = tension_hi/100.0;
extr_hi = 100.0*(xx+fa[i_hi][j_hi]*xx/(1.0+fb[i_hi][j_hi]*xx)+
              fc[i_hi][j_hi]*pow(xx,4.0)/(1.0+
              fd[i_hi][j_hi]*pow(xx,4.0)));

xx = tension_lo/100.0;
extr_lo = 100.0*(xx+fa[i_lo][j_lo]*xx/(1.0+fb[i_lo][j_lo]*xx)+
              fc[i_lo][j_lo]*pow(xx,4.0)/(1.0+
              fd[i_lo][j_lo]*pow(xx,4.0)));
```

```
xx = mean/100.0;
ext = 100.0*(xx+fa[i][j]*xx/(1.0+fb[i][j]*xx)+
             fc[i][j]*pow(xx,4.0)/(1.0+
             fd[i][j]*pow(xx,4.0)));

/* save results */
spd_data[0][0]=spdlo;
spd_data[1][0]=oldspd;
spd_data[2][0]=spdhi;
spd_data[0][1]=curve_lo;
spd_data[1][1]=curve;
spd_data[2][1]=curve_hi;
spd_data[0][2]=tension_lo;
spd_data[1][2]=mean;
spd_data[2][2]=tension_hi;
spd_data[0][3]=extr_lo;
spd_data[1][3]=ext;
spd_data[2][3]=extr_hi;

/* print results on screen; give option to graph */
clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);
write_header(startrow,startcol,endcol,label1[0],REV_VID);

/* -- write labels */
write_string(startrow+5,startcol+2,label1[6],REV_VID);
write_string(startrow+7,startcol+2,label1[10],REV_VID);
write_string(startrow+8,startcol+2,label1[11],REV_VID);
write_string(startrow+9,startcol+2,label1[12],REV_VID);
write_string(startrow+10,startcol+2,label1[13],REV_VID);

/* -- write tow speeds */
sprintf(labx,"%6.1f",spdlo);
start=startcol+2+strlen(label1[13])+1;
write_string(startrow+5,start,labx,REV_VID);
sprintf(labx,"%6.1f",oldspd);
start=startcol+2+strlen(label1[13])+1+7;
write_string(startrow+5,start,labx,REV_VID);
sprintf(labx,"%6.1f",spdhi);
start=startcol+2+strlen(label1[13])+1+14;
write_string(startrow+5,start,labx,REV_VID);

/* -- write curve numbers */
sprintf(labx,"%d",curve_lo);
start=startcol+2+strlen(label1[13])+1+3;
write_string(startrow+7,start,labx,REV_VID);
sprintf(labx,"%d",curve);
start=startcol+2+strlen(label1[13])+1+7+3;
write_string(startrow+7,start,labx,REV_VID);
sprintf(labx,"%d",curve_hi);
start=startcol+2+strlen(label1[13])+1+14+3;
write_string(startrow+7,start,labx,REV_VID);

/* -- write mean tensions */
```

```
sprintf(labx,"%6.1f",tension_lo);
start=startcol+2+strlen(label1[13])+1;
write_string(startrow+8,start,labx,REV_VID);
sprintf(labx,"%6.1f",mean);
start=startcol+2+strlen(label1[13])+1+7;
write_string(startrow+8,start,labx,REV_VID);
sprintf(labx,"%6.1f",tension_hi);
start=startcol+2+strlen(label1[13])+1+14;
write_string(startrow+8,start,labx,REV_VID);

/* -- write dynamic tensions */
xx = extr_lo - tension_lo;
if (xx<0) xx=0;
sprintf(labx,"%6.1f",xx);
start=startcol+2+strlen(label1[13])+1;
write_string(startrow+9,start,labx,REV_VID);
xx = ext - mean;
if (xx<0) xx=0;
sprintf(labx,"%6.1f",xx);
start=startcol+2+strlen(label1[13])+1+7;
write_string(startrow+9,start,labx,REV_VID);
xx = extr_hi - tension_hi;
if (xx<0) xx=0;
sprintf(labx,"%6.1f",xx);
start=startcol+2+strlen(label1[13])+1+14;
write_string(startrow+9,start,labx,REV_VID);

/* -- write extreme tensions */
sprintf(labx,"%6.1f",extr_lo);
start=startcol+2+strlen(label1[13])+1;
write_string(startrow+10,start,labx,REV_VID);
sprintf(labx,"%6.1f",ext);
start=startcol+2+strlen(label1[13])+1+7;
write_string(startrow+10,start,labx,REV_VID);
sprintf(labx,"%6.1f",extr_hi);
start=startcol+2+strlen(label1[13])+1+14;
write_string(startrow+10,start,labx,REV_VID);

/* -- give menu to graph or quit */
start = (endcol-startcol-strlen(menu2[0]))/2 + startcol;
choice=popup(menu2,"12",2,startrow+12,start-2,NONE,REV_VID,0);

if (choice==1)
  return;
else if (choice==0)
{
  setplt();                          /* initialize graphics */
  locate(&x0i,&y0i,&xli,&yli);       /* set position on screen */
  xyaxis(&xmn,&xmx,&xtc,&ymn,&ymx,
              &ytc,&xmn,&ymn,&kd);   /* draw axes */
  axnum(&ntx,&nty);                  /* write axis numbers */
  itype=1;
  sprintf(labx,"Mean Tension (kips)_");
  label(&itype,labx);                /* write x-axis label */
```

```
itype=2;
sprintf(laby,"Extreme Tension (kips)_");
label(&itype,laby);                    /* write y-axis label */

npts=122;
line3(&npts,x_hi,y_hi);                /* draw line thru pts */
line3(&npts,x_lo,y_lo);
line3(&npts,x,y);

/* if mean > 120 kips, don't plot points */
if (tension_lo<=120.0)
{
  npts=1;                              /* plot symbols       */
  x_lo[0]= tension_lo;
  y_lo[0]= extr_lo;
  nsym=4;
  symplt(&npts,x_lo,y_lo,&nsym);
}
x_lo[0]= 30.0;
y_lo[0]= ysm+5;
symplt(&npts,x_lo,y_lo,&nsym);

if (mean<=120.0)
{
  x[0]= mean;
  y[0]= ext;
  nsym=3;
  symplt(&npts,x,y,&nsym);
}
x[0]= 55.0;
y[0]= ysm+5;
symplt(&npts,x,y,&nsym);

if (tension_hi<=120.0)
{
  x_hi[0]= tension_hi;
  y_hi[0]= extr_hi;
  nsym=2;
  symplt(&npts,x_hi,y_hi,&nsym);
}
x_hi[0]= 80.0;
y_hi[0]= ysm+5;
symplt(&npts,x_hi,y_hi,&nsym);

/* write labels */
x1= 35.0;
y1= ysm;
sprintf(labx,"%-.0f kts_",spdlo);
labely(&x1,&y1,labx);
x1= 60.0;
y1= ysm;
sprintf(labx,"%-.0f kts_",oldspd);
labely(&x1,&y1,labx);
x1= 85.0;
```

```
    y1= ysm;
    sprintf(labx,"%-.0f kts_",spdhi);
    labely(&x1,&y1,labx);

    wait_plt();
  )
  else
    return ;
}

/*************************************************************************
     This function shows the effects of hawser length on extreme tension
  by computing the mean tension for Length +/- 300 feet, finding the
  appropriate extreme tension curves, and displaying a graph of all three
  curves. A summary of results is displayed prior to displaying the graphs.
*************************************************************************/
void show_length(type,i,j,curve)
int type;
int i,j,curve;
{
  float x_hi[121],x_lo[121],y_hi[121],y_lo[121];
  float x[121],y[121];
  float tension_hi,tension_lo,extr_hi,extr_lo;
  float mean,ext;
  float hilgth,lolgth,oldlgth;
  int tugtype,towtype,tugtow,windspeed,head,towspeed;
  int lengthhi,lengthlo;
  int curve_hi,i_hi,j_hi;
  int curve_lo,i_lo,j_lo;
  int k;                        /* dummy counter */
  float x1,y1,x2,y2;            /* coordinates for drawing */
  char labx[25],laby[25];       /* strings for x, y axis labels */
  short kd=2;                   /* graph "kode" */
  short npts;                   /* number of points to plot */
  short ntx=2,nty=1;            /* skip factor for axis numbers */
  short itype;                  /* designates x or y axis label in label() */
  short nsym;                   /* symbol code */
  float x0i=1.0764;
  float y0i=1.0;
  float xli=6.624;
  float yli=4.35;
  float xmn=0.0;                /* min x value */
  float xmx=120.0;              /* max x value */
  float xtc=10.0;               /* x axis tic value */
  float ymn=0.0;                /* min y value */
  float ymx=400.0;              /* max y value */
  float ytc=50.0;               /* y axis tic value */
  float xxmax;                  /* x corresponding to y=400.0 kips */
  float ;
  float res_dat[5];
  float tow_res,haw_res;
  float xx;
  float xsm,ysm;
  int start,choice;
```

```
xsm = xmn - (x0i/xli)*(xmx-xmn);
ysm = ymn - (y0i/yli)*(ymx-ymn);

/* write message */
draw_window(startrow+11,startcol+12,startrow+13,startcol+27,
              SINGLE,REV_VID);
sprintf(labx,"Please wait");
start = (endcol-startcol-strlen(labx))/2 + startcol;
write_string(startrow+12,start,labx,BLINK_REV_VID);

/* set hawser lengths */
oldlgth=tug_data[2];
lolgth=tug_data[2]-300.0;
if (lolgth<1000.0)
  lolgth=1000.0;
hilgth=tug_data[2]+300.0;
if (hilgth>2100.0)
  hilgth=2100.0;

/* compute mean tension */
if (type==SHIP)
{
  ship_resist(tow_data,ship_data,res_dat,&tow_res);

  /* -- for longer length */
  tug_data[2]=hilgth;
  hawser_resist(tug_data,tow_data[1],tow_res,&haw_res);
  tension_hi = tow_res + haw_res;

  /* -- for shorter length */
  tug_data[2]=lolgth;
  hawser_resist(tug_data,tow_data[1],tow_res,&haw_res);
  tension_lo = tow_res + haw_res;

  /* -- for given length */
  tug_data[2]=oldlgth;
  hawser_resist(tug_data,tow_data[1],tow_res,&haw_res);
  mean = tow_res + haw_res;
}
else if (type==DOCK)
{
  /* -- for longer length */
  tug_data[2]=hilgth;
  dock_resist(tug_data,dock_data,tow_data,&tension_hi);

  /* -- for shorter length */
  tug_data[2]=lolgth;
  dock_resist(tug_data,dock_data,tow_data,&tension_lo);

  /* -- for given length */
  tug_data[2]=oldlgth;
  dock_resist(tug_data,dock_data,tow_data,&mean);
}
```

```
else          /* type==BARGE */
{
  /* -- for longer length */
  tug_data[2]=hilgth;
  barge_resist(tug_data,barge_data,tow_data,&tension_hi);

  /* -- for shorter length */
  tug_data[2]=lolgth;
  barge_resist(tug_data,barge_data,tow_data,&tension_lo);

  /* -- for given length */
  tug_data[2]=oldlgth;
  barge_resist(tug_data,barge_data,tow_data,&mean);
}

/* convert to kips */
tension_lo /= 1000.0;
tension_hi /= 1000.0;
mean /= 1000.0;

/* set common indices */
set_tugtype(&tugtype);
set_towtype(&towtype);
tugtow = towtype + 6*tugtype;

set_towspd(tow_data[1],&towspeed);
set_wind(tow_data[2],&windspeed);
set_hdg(tow_data[3],&head);

/* find curve for hilgth */
set_scope(hilgth,&lengthhi);
curve_hi=t[tugtow].s[windspeed][towspeed].c[head][lengthhi];
curvindex(curve_hi,&i_hi,&j_hi);

/* find curve for lolgth */
set_scope(lolgth,&lengthlo);
curve_lo=t[tugtow].s[windspeed][towspeed].c[head][lengthlo];
curvindex(curve_lo,&i_lo,&j_lo);

/* compute tension arrays */
comp_ten_curv(i_hi,j_hi,x_hi,y_hi);
clip(x_hi,y_hi);
comp_ten_curv(i_lo,j_lo,x_lo,y_lo);
clip(x_lo,y_lo);
comp_ten_curv(i,j,x,y);
clip(x,y);

/* compute extremes */
xx = tension_hi/100.0;
extr_hi = 100.0*(xx+fa[i_hi][j_hi]*xx/(1.0+fb[i_hi][j_hi]*xx)+
              fc[i_hi][j_hi]*pow(xx,4.0)/(1.0+
              fd[i_hi][j_hi]*pow(xx,4.0)));

xx = tension_lo/100.0;
```

```
extr_lo = 100.0*(xx+fa[i_lo][j_lo]*xx/(1.0+fb[i_lo][j_lo]*xx)+
              fc[i_lo][j_lo]*pow(xx,4.0)/(1.0+
              fd[i_lo][j_lo]*pow(xx,4.0)));

xx = mean/100.0;
ext = 100.0*(xx+fa[i][j]*xx/(1.0+fb[i][j]*xx)+
              fc[i][j]*pow(xx,4.0)/(1.0+
              fd[i][j]*pow(xx,4.0)));

/* save results */
lgth_data[0][0]=lolgth;
lgth_data[1][0]=oldlgth;
lgth_data[2][0]=hilgth;
lgth_data[0][1]=curve_lo;
lgth_data[1][1]=curve;
lgth_data[2][1]=curve_hi;
lgth_data[0][2]=tension_lo;
lgth_data[1][2]=mean;
lgth_data[2][2]=tension_hi;
lgth_data[0][3]=extr_lo;
lgth_data[1][3]=ext;
lgth_data[2][3]=extr_hi;

/* print results on screen; give option to graph */
clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);
write_header(startrow,startcol,endcol,label1[0],REV_VID);

/* -- write labels */
write_string(startrow+5,startcol+2,label1[9],REV_VID);
write_string(startrow+7,startcol+2,label1[10],REV_VID);
write_string(startrow+8,startcol+2,label1[11],REV_VID);
write_string(startrow+9,startcol+2,label1[12],REV_VID);
write_string(startrow+10,startcol+2,label1[13],REV_VID);

/* -- write hawser lengths */
sprintf(labx,"%6.0f",lolgth);
start=startcol+2+strlen(label1[13])+1;
write_string(startrow+5,start,labx,REV_VID);
sprintf(labx,"%6.0f",oldlgth);
start=startcol+2+strlen(label1[13])+1+7;
write_string(startrow+5,start,labx,REV_VID);
sprintf(labx,"%6.0f",hilgth);
start=startcol+2+strlen(label1[13])+1+14;
write_string(startrow+5,start,labx,REV_VID);

/* -- write curve numbers */
sprintf(labx,"%d",curve_lo);
start=startcol+2+strlen(label1[13])+1+3;
write_string(startrow+7,start,labx,REV_VID);
sprintf(labx,"%d",curve);
start=startcol+2+strlen(label1[13])+1+7+3;
write_string(startrow+7,start,labx,REV_VID);
sprintf(labx,"%d",curve_hi);
start=startcol+2+strlen(label1[13])+1+14+3;
```

```
write_string(startrow+7,start,labx,REV_VID);

/* -- write mean tensions */
sprintf(labx,"%6.1f",tension_lo);
start=startcol+2+strlen(label1[13])+1;
write_string(startrow+8,start,labx,REV_VID);
sprintf(labx,"%6.1f",mean);
start=startcol+2+strlen(label1[13])+1+7;
write_string(startrow+8,start,labx,REV_VID);
sprintf(labx,"%6.1f",tension_hi);
start=startcol+2+strlen(label1[13])+1+14;
write_string(startrow+8,start,labx,REV_VID);

/* -- write dynamic tensions */
xx = extr_lo - tension_lo;
if (xx<0) xx=0;
sprintf(labx,"%6.1f",xx);
start=startcol+2+strlen(label1[13])+1;
write_string(startrow+9,start,labx,REV_VID);
xx = ext - mean;
if (xx<0) xx=0;
sprintf(labx,"%6.1f",xx);
start=startcol+2+strlen(label1[13])+1+7;
write_string(startrow+9,start,labx,REV_VID);
xx = extr_hi - tension_hi;
if (xx<0) xx=0;
sprintf(labx,"%6.1f",xx);
start=startcol+2+strlen(label1[13])+1+14;
write_string(startrow+9,start,labx,REV_VID);

/* -- write extreme tensions */
sprintf(labx,"%6.1f",extr_lo);
start=startcol+2+strlen(label1[13])+1;
write_string(startrow+10,start,labx,REV_VID);
sprintf(labx,"%6.1f",ext);
start=startcol+2+strlen(label1[13])+1+7;
write_string(startrow+10,start,labx,REV_VID);
sprintf(labx,"%6.1f",extr_hi);
start=startcol+2+strlen(label1[13])+1+14;
write_string(startrow+10,start,labx,REV_VID);

/* -- give menu to graph or quit */
start = (endcol-startcol-strlen(menu2[0]))/2 + startcol;
choice=popup(menu2,"12",2,startrow+12,start-2,NONE,REV_VID,0);

if (choice==1)
  return;
else if (choice==0)
{
  setplt();                              /* initialize graphics */
  locate(&x0i,&y0i,&xli,&yli);           /* set position on screen */
  xyaxis(&xmn,&xmx,&xtc,&ymn,&ymx,&ytc,&xmn,&ymn,&kd);  /* draw axes */
  axnum(&ntx,&nty);                      /* write axis numbers */
  itype=1;
```

```
sprintf(labx,"Mean Tension (kips)_");
label(&itype,labx);                    /* write x-axis label */
itype=2;
sprintf(laby,"Extreme Tension (kips)_");
label(&itype,laby);                    /* write y-axis label */

npts=122;
line3(&npts,x_hi,y_hi);                /* draw line thru pts */
line3(&npts,x_lo,y_lo);                /*        "           */
line3(&npts,x,y);                      /*        "           */

/* don't plot points if mean tension > 120 kips */
if (tension_lo<=120.0)
{
  npts=1;                              /* plot symbols       */
  x_lo[0]= tension_lo;
  y_lo[0]= extr_lo;
  nsym=4;
  symplt(&npts,x_lo,y_lo,&nsym);
}
x_lo[0]= 30.0;
y_lo[0]= ysm+5;
symplt(&npts,x_lo,y_lo,&nsym);

if (mean<=120.0)
{
  x[0]= mean;
  y[0]= ext;
  nsym=3;
  symplt(&npts,x,y,&nsym);
}
x[0]= 55.0;
y[0]= ysm+5;
symplt(&npts,x,y,&nsym);

if (tension_lo<=120.0)
{
  x_hi[0]= tension_hi;
  y_hi[0]= extr_hi;
  nsym=2;
  symplt(&npts,x_hi,y_hi,&nsym);
}
x_hi[0]= 80.0;
y_hi[0]= ysm+5;
symplt(&npts,x_hi,y_hi,&nsym);

/* write labels */
x1= 35.0;
y1= ysm;
sprintf(labx,"%-.0f ft_",lolgth);
labely(&x1,&y1,labx);
x1= 60.0;
y1= ysm;
sprintf(labx,"%-.0f ft_",oldlgth);
```

```
    labely(&x1,&y1,labx);
    x1= 85.0;
    y1= ysm;
    sprintf(labx,"%-.0f ft_",hilgth);
    labely(&x1,&y1,labx);

    wait_plt();
  }
  else
    return ;
}

/*************************************************************
  This function computes the array of points to plot for
  the standard tension curve indicated by i and j.
*************************************************************/
void comp_ten_curv(i,j,x,y)
int i,j;
float *x,*y;
{
  int k;
  float x1;

  for (k=0; k<121; k++)
  {
    x[k]=(float)k;
    x1=x[k]/100.0;
    y[k]= 100.0*(x1+fa[i][j]*x1/(1.0+fb[i][j]*x1)
           + fc[i][j]*x1*x1*x1*x1/(1.0+fd[i][j]*x1*x1*x1*x1));
  }
}

/*************************************************************
  This function finds the intersection of a plotted curve
  with the top border and "clips" it to keep everything
  within the borders of the graph.
*************************************************************/
void clip(x,y)
float *x,*y;
{
  int flag=0;
  int k;
  float x1,x2,y1,y2,xxmax;

  /* test for extreme > 400 kips */
  for (k=0; k<121; k++)
  {
    if (y[k]<=400.0)
    {
      x1=x[k];
      y1=y[k];
    }
    else if (!flag && y[k]>400.0)
    {
```

```
      x2=x[k];
      y2=y[k];
      flag=1;

      /* interpolate to find xxmax */
      xxmax=x1+(x2-x1)/(y2-y1)*(400.0 - y1);

      /* set next plotting point to intersection of curve with border */
      y[k]=400.0;
      x[k]=xxmax;
    }
    else
    {
      y[k]=400.0;
    }
  }
}

/*************************************************************
  This function pauses until the user hits a key, then
  closes the graphics environment.
*************************************************************/
void wait_plt()
{
  union inkey {
    char ch[2];
    int i;
  } c;

  while (1)
  {
    c.i = bioskey(0);        /* read the key */

    if (c.ch[0])             /* key is a normal key */
      break;
    else                     /* key is a special key */
      break;
  }
  endplt();                          /* close graphics     */
}
```

```
/*****************************************************************************
  File: pull.c
  Author: Todd J. Peltzer
  Last update: 24 April 1989

    This file contains the functions which evaluate a tug's ability to tow
  a given vessel at a specified speed.
-----------------------------------------------------------------------------
  Functions:
    tugpull()
    show_tugpull()
    find_avail()
    get_tugpull_opt()
    tugpull_scrn()
    tugpull_data()
*****************************************************************************/
#include "stdio.h"
#include "keydef.h"
#include "video.h"

void tugpull(),show_tugpull(),tugpull_scrn(),tugpull_data();
float find_avail();

extern char tug[15];            /* tug class                        */
extern char hull_no[24];        /* hull number entered by user      */
extern float tug_data[5];       /* array to store tug data          */
extern float tow_data[5];       /* array to store tow data          */
extern float ship_data[6];      /* array to store Table G-2 data    */
extern float dock_data[7];      /* array to store drydock data      */
extern float barge_data[8];     /* array to store barge data        */
extern float tug_eval[7];       /* tug evaluation results           */

static int startrow=2;
static int startcol=20;
static int endrow=22;
static int endcol=60;

static char *menu[] =
{
  "1) Use best possible speed ",
  "2) Use original tow speed  ",
  "3) Show graph              ",
  "4) Return to PROGRAM OPTIONS"
};

static char *label1[] =
{
  "TUG EVALUATION",
  "Tug:",
  "Tow:",
  "Desired tow speed:",
  "Mean tension:",
  "Available tension:",
  "Best tow speed:",
```

```
  "Please wait"
};

/*************************************************************************
  This function compares the mean towline tension with the available
  towline tension for the given tug at the specified tow speed. If there
  is insufficient tension available, the tow speed is reduced until the
  available tension exceeds the mean towline tension.
*************************************************************************/
void tugpull(type,tension)
int type;                       /* type of tow: ship, dock, or barge */
float *tension;                 /* mean towline tension */
{
  FILE *in;                       /* pointer to input file           */
  int flag=0;                     /* status flag                     */
  int i,j;                        /* counters                        */
  int tugtype;                    /* type of tug                     */
  char inline[81];                /* string for getting input        */
  char text1[37],text2[37];       /* error message text strings      */
  char string[25];                /* text string                     */
  float array[5];                 /* input data array                */
  float status,nbr;               /* required arguments for stofa()  */
  float speed[20];                /* array of tow speeds             */
  float avail[4][20];             /* array of available tensions     */
  float avail_ten;                /* return value of find_avail()    */
  float old_avail;                /* value used in iteration scheme  */
  float vtow_avail;               /* available tension at vtow       */
  float best_avail;               /* available tension at best speed */
  float old_mean;                 /* value used in iteration scheme  */
  float vtow;                     /* tow speed                       */
  float mean = *tension;          /* tension at original tow speed   */
  float best_mean;                /* tension at best speed           */
  float best_spd;                 /* best possible speed             */
  float res_dat[5];               /* required arg. for ship_resist() */
  float tow_res,haw_res;          /* tow, hawser resistance          */
  int choice;                     /* popup menu selection            */
  int start,hilite=0;

  in=fopen("tugpull.dat","r");

  vtow=tug_eval[0]=tow_data[1];
  tug_eval[1] = *tension;

  /* draw screen, labels, preliminary data */
  tugpull_scrn(vtow,mean);

  /* read available tow tension data from file */
  fgets(inline,81,in);
  fgets(inline,81,in);
  for (i=0; i<20; i++)
  {
    fgets(inline,81,in);
    status = stofa(inline,array,&nbr,5);
    speed[i]=array[0];
```

```
  for (j=0; j<4; j++)
  {
    avail[j][i]=array[j+1];
  }
}

/* find available tow tension for given tow speed */
tugtype=tug_data[0];
avail_ten=find_avail(tugtype,vtow,speed,avail);
vtow_avail=tug_eval[2]=avail_ten;
if (avail_ten < 0.0)
{
  sprintf(text1,"Tow speed outside the limits of");
  sprintf(text1,"Figure 6-1");
  display_error(ERROR,text1,text2);
  get_options(1);
  return;
}

/* write available tension */
start=startcol+3+strlen(label1[3])+5;
sprintf(string,"%6.0f",avail_ten);
write_string(startrow+9,start,string,REV_VID);

/* compare mean tension to available tension */
i=0;
old_avail=avail_ten;
old_mean = *tension;
while(1)
{
  best_spd=tow_data[1]=speed[i];
  if (type==SHIP)
  {
    ship_resist(tow_data,ship_data,res_dat,&tow_res);
    hawser_resist(tug_data,tow_data[1],tow_res,&haw_res);
    *tension = tow_res+haw_res;
  }
  else if (type==DOCK)
  {
    dock_resist(tug_data,dock_data,tow_data,tension);
  }
  else          /* type==BARGE */
  {
    barge_resist(tug_data,barge_data,tow_data,tension);
  }

  avail_ten=find_avail(tugtype,best_spd,speed,avail);

  if (avail_ten < 0.0)
  {
    sprintf(text1,"Tow speed outside the limits of");
    sprintf(text2,"Figure 6-1");
    display_error(ERROR,text1,text2);
  }
```

```
  else if (avail_ten < *tension)
  {
    best_spd=speed[i-1];
    best_avail=avail_ten=old_avail;
    best_mean = *tension = old_mean;
    break;
  }
  else
  {
    old_avail=avail_ten;
    old_mean=*tension;
    i++;
  }
)

/* erase wait message */
clear(startrow+15,startcol+12,startrow+17,startcol+27,REV_VID);

/* write best possible speed */
start=startcol+3+strlen(label1[3])+5;
sprintf(string,"%6.1f",best_spd);
write_string(startrow+11,start,string,REV_VID);

/* write mean tension */
sprintf(string,"%6.0f",*tension);
write_string(startrow+12,start,string,REV_VID);

/* write available tension */
sprintf(string,"%6.0f",best_avail);
write_string(startrow+13,start,string,REV_VID);

/* write units */
start=endcol-5;
sprintf(string,"kts");
write_string(startrow+11,start,string,REV_VID);
sprintf(string,"lbs");
write_string(startrow+12,start,string,REV_VID);
write_string(startrow+13,start,string,REV_VID);

/* save results */
tug_eval[3]=best_spd;
tug_eval[4]=best_mean;
tug_eval[5]=best_avail;

/* display options, get response */
while (1)
{
  choice=get_tugpull_opt(hilite);

  if (choice==0)        /* use best possible speed */
  {
    tow_data[1]=best_spd;
    *tension=best_mean;
    tug_eval[6]=0;
```

```
  hilite=2;
  flag=1;
  sprintf(text1,"Best possible tow speed set");
  display_error(BLANK,text1," ");
  cursor_off();
  continue;
}
else if (choice==1)   /* use original tow speed */
{
  if (vtow_avail<mean)       /* can't make desired speed */
  {
    sprintf(text1,"Tug has inadequate pull at this speed");
    sprintf(text2,"==> choose best speed option <==");
    display_error(ERROR,text1,text2);
    hilite=0;
    flag=0;
    cursor_off();
    continue;
  }
  tow_data[1]=vtow;
  *tension=mean;
  tug_eval[6]=1;
  hilite=2;
  flag=1;
  sprintf(text1,"Original tow speed set");
  display_error(BLANK,text1," ");
  cursor_off();
}
else if (choice==2)   /* show graph */
{
  if (!flag)          /* speed not chosen */
  {
    sprintf(text1,"Tow speed has not been selected");
    sprintf(text2,"==> please choose a tow speed <==");
    display_error(ERROR,text1,text2);
    hilite=0;
    cursor_off();
    continue;
  }
  show_tugpull(speed,avail,vtow,best_spd,vtow_avail,best_avail);
  hilite=3;
  cursor_off();
  tugpull_scrn(vtow,mean);
  tugpull_data(vtow_avail,best_spd,best_mean,best_avail);
  continue;
}
else if (choice==3 || choice<0)
{
  if (!flag)          /* speed not chosen */
  {
    sprintf(text1,"Tow speed has not been selected");
    sprintf(text2,"==> please choose a tow speed <==");
    display_error(ERROR,text1,text2);
    hilite=0;
```

```
        cursor_off();
        continue;
      }
      break;
    }
  }
  cursor_off();
  return;
}

/*****************************************************************************
  This function finds the available tension from the given tug at the
  given speed.
*****************************************************************************/
float find_avail(tug,vtow,speed,avail)
int tug;
float vtow;
float *speed;
float avail[][20];
{
  int i;
  float avail_ten;
  float delx,fac;

  for (i=0; i<20; i++)
  {
    if (vtow==speed[i])
    {
      avail_ten=avail[tug][i];
      return avail_ten;
    }
    else if (vtow<speed[i])
      continue;
    else if (vtow>speed[i] && vtow<speed[i+1])
    {
      delx=speed[i+1] - speed[i];
      fac=(vtow-speed[i])/delx;
      avail_ten=(avail[tug][i+1]-avail[tug][i])*fac+avail[tug][i];
      return avail_ten;
    }
  }
  return -1.0;
}

/*****************************************************************************
  This function displays the options available and returns the user's
  choice.
*****************************************************************************/
get_tugpull_opt(hilite)
int hilite;
{
  int choice,start;

  /* write options menu */
```

```
  start=(endcol-startcol-strlen(menu[0]))/2 + startcol;
  choice=popup(menu,"1234",4,startrow+14,start-2,NONE,REV_VID,hilite);

  return choice;
}

/*************************************************************************
  This function displays a reproduction of Figure 6-1 in the Towing
  Manual, showing available tension versus tow speed for each class
  of tug used in this program (ATF is not used);
*************************************************************************/
void show_tugpull(speed,avail,vtow,best_spd,vtow_avail,best_avail)
float *speed;
float avail[][20];
float vtow,best_spd;
float vtow_avail,best_avail;
{
  int j,k;                    /* dummy counter */
  float x[101],y[101];        /* arrays to hold x, y coordinates */
  float x1,y1,x2,y2;          /* coordinates for drawing lines */
  char labx[25],laby[25];     /* strings for x, y axis labels */
  short kd=2;                 /* graph "kode" */
  short npts;                 /* number of points to plot */
  short ntx=2,nty=2;          /* skip factor for axis numbers */
  short itype;                /* designates x or y axis label in label() */
  short nsym;                 /* symbol code */
  float x0i=1.0764;           /* screen parameters */
  float y0i=1.0;              /*    "       "      */
  float xli=6.624;            /*    "       "      */
  float yli=4.35;             /*    "       "      */
  float xmn=0.0;              /* min x value */
  float xmx=16.0;             /* max x value */
  float xtc=2.0;              /* x axis tic value */
  float ymn=0.0;              /* min y value */
  float ymx=200.0;            /* max y value */
  float ytc=25.0;             /* y axis tic value */
  float xxmax;
  float xsm,ysm;              /* smallest x,y in world coordinates */

  xsm = xmn - (x0i/xli)*(xmx-xmn);
  ysm = ymn - (y0i/yli)*(ymx-ymn);

  setplt();                           /* initialize graphics */
  locate(&x0i,&y0i,&xli,&yli);        /* set position on screen */
  xyaxis(&xmn,&xmx,&xtc,&ymn,&ymx,
        &ytc,&xmn,&ymn,&kd);          /* draw axes */
  axnum(&ntx,&nty);                   /* write axis numbers */
  itype=1;
  sprintf(labx,"Tow Speed (kts)_");
  label(&itype,labx);                 /* write x-axis label */
  itype=2;
  sprintf(laby,"Available Tension (kips)_");
  label(&itype,laby);                   /* write y-axis label */
```

```
npts=21;
for (k=0; k<4; k++)
{
  for (j=0; j<20; j++)
  {
    x[j]=speed[j];
    y[j]=avail[k][j]/1000.0;
  }
  line3(&npts,x,y);                    /* draw line thru pts */
)

x1= speed[4];                          /* write labels */
y1= avail[3][4]/1000.0+5.0;
sprintf(labx,"T-ATF 166_");
labely(&x1,&y1,labx);

x1= speed[2];
y1= avail[1][2]/1000.0-20.0;
sprintf(labx,"ARS 50_");
labely(&x1,&y1,labx);

x1= speed[12];
y1= avail[2][12]/1000.0+5.0;
sprintf(labx,"ATS 1_");
labely(&x1,&y1,labx);

x1= speed[2];
y1= avail[0][2]/1000.0+5.0;
sprintf(labx,"ARS 38_");
labely(&x1,&y1,labx);

/* plot symbols */
npts=1;
x1= vtow;
y1= vtow_avail/1000.0;
nsym=4;
symplt(&npts,&x1,&y1,&nsym);
x1= 3.0;
y1= ysm+2.5;
symplt(&npts,&x1,&y1,&nsym);
x1= best_spd;
y1= best_avail/1000.0;
nsym=2;
symplt(&npts,&x1,&y1,&nsym);
x1= 10.0;
y1= ysm+2.5;
symplt(&npts,&x1,&y1,&nsym);

/* write labels */
x1= 4.0;
y1= ysm;
sprintf(labx,"Desired Speed");
labely(&x1,&y1,labx);
x1= 11.0;
```

```
  y1= ysm;
  sprintf(labx,"Best Speed");
  labely(&x1,&y1,labx);

  /* wait for keystroke, then exit */
  wait_plt();

}

/*************************************************************************
  This function creates the tug evaluation display.
*************************************************************************/
void tugpull_scrn(vtow,tension)
float vtow,tension;
{
  int i,start;
  char string[25];

  draw_window(startrow,startcol,endrow,endcol,DOUBLE,REV_VID);

  /* write header */
  write_header(startrow,startcol,endcol,label1[0],REV_VID);

  /* write labels */
  for (i=1; i<6; i++)
    write_string(startrow+4+i,startcol+3,label1[i],REV_VID);

  /* draw dividing line */
  for (i=0; i<(endcol-startcol-5); i++)
    write_char(startrow+10,startcol+3+i,196,REV_VID);

  /* write remaining labels */
  write_string(startrow+11,startcol+3,label1[6],REV_VID);
  write_string(startrow+12,startcol+3,label1[4],REV_VID);
  write_string(startrow+13,startcol+3,label1[5],REV_VID);

  /* write tug and tow */
  start=startcol+3+strlen(label1[1])+2;
  write_string(startrow+5,start,tug,REV_VID);
  write_string(startrow+6,start,hull_no,REV_VID);

  /* write desired tow speed, mean tension */
  start=startcol+3+strlen(label1[3])+5;
  sprintf(string,"%6.1f",vtow);
  write_string(startrow+7,start,string,REV_VID);
  sprintf(string,"%6.0f",tension);
  write_string(startrow+8,start,string,REV_VID);

  /* write units */
  start=endcol-5;
  sprintf(string,"kts");
  write_string(startrow+7,start,string,REV_VID);
  sprintf(string,"lbs");
  write_string(startrow+8,start,string,REV_VID);
```

```
  write_string(startrow+9,start,string,REV_VID);

  /* write wait message */
  draw_window(startrow+15,startcol+12,startrow+17,startcol+27,
              SINGLE,REV_VID);
  sprintf(string,"Please wait");
  start = (endcol-startcol-strlen(string))/2 + startcol;
  write_string(startrow+16,start,string,BLINK_REV_VID);
  return;
}

/*****************************************************************************
  This function writes the results of the tug evaluation on the screen.
*****************************************************************************/
void tugpull_data(vtow_avail,best_spd,best_mean,best_avail)
float vtow_avail;
float best_spd;
float best_mean;
float best_avail;
{
  int start;
  char string[25];

  /* erase wait message */
  clear(startrow+15,startcol+12,startrow+17,startcol+27,REV_VID);

  /* write available tension */
  start=startcol+3+strlen(label1[3])+5;
  sprintf(string,"%6.0f",vtow_avail);
  write_string(startrow+9,start,string,REV_VID);

  /* write best possible speed */
  start=startcol+3+strlen(label1[3])+5;
  sprintf(string,"%6.1f",best_spd);
  write_string(startrow+11,start,string,REV_VID);

  /* write mean tension */
  sprintf(string,"%6.0f",best_mean);
  write_string(startrow+12,start,string,REV_VID);

  /* write available tension */
  sprintf(string,"%6.0f",best_avail);
  write_string(startrow+13,start,string,REV_VID);

  /* write units */
  start=endcol-5;
  sprintf(string,"kts");
  write_string(startrow+11,start,string,REV_VID);
  sprintf(string,"lbs");
  write_string(startrow+12,start,string,REV_VID);
  write_string(startrow+13,start,string,REV_VID);
  return;
}
```

```
/*************************************************************************
  File: rp.c
  Author: Todd J. Peltzer
  Last update: 19 April 1989

    This file contains the functions which generate a report and send
  it to the printer
-------------------------------------------------------------------------
  Functions:
    report()
*************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "string.h"
#include "video.h"

void report();

static char *tabtow[] =
{
  "YRBM",
  "FFG 1",
  "DD 963",
  "AE 26",
  "LHA 1",
  "CVN 65"
};

extern char tug[15];            /* tug class                        */
extern char hull_no[24];        /* hull number entered by user      */
extern char class[24];          /* class of ship from Table G-2     */
extern float tug_data[5];       /* array to store tug data          */
extern float tow_data[5];       /* array to store tow data          */
extern float ship_data[6];      /* array to store Table G-2 data    */
extern float dock_data[7];      /* array to store drydock data      */
extern float barge_data[8];     /* array to store barge data        */
extern float tension;           /* mean towline tension             */
extern float extr_ten;          /* extreme towline tension          */
extern int curve;               /* curve number from extreme        */
extern int flag[3];             /* error checking flag array        */
extern int type;                /* tow type                         */
extern float barge_res[6];      /* computed barge parameters        */
extern float resist_dat[5];        /* resistance data                  */
extern float ext_data[10];      /* extreme tension results          */
extern float spd_data[3][4];    /* tow speed effects results        */
extern float lgth_data[3][4];   /* hawser length effects results    */
extern float tug_eval[7];

static char *menu[] =
{
  "1) Send report to printer   ",
  "2) Send report to file only ",
  "3) Return to PROGRAM OPTIONS"
};
```

```
/*************************************************************************
    This function generates the report, and sends it to the printer
  if the user has indicated that choice.
*************************************************************************/
void report()
{
  char inline[81],outline[81],text[40];
  char text1[39],text2[39];
  char mssg[9][81];
  int start,end,i,j,choice,prnt=0;
  float tow_res,haw_res;
  float x1,x2,x3;
  FILE *in,*out;
  int startrow=2;
  int startcol=20;
  int endrow=22;
  int endcol=60;

  clear(startrow+1,startcol+1,endrow-1,endcol-1,REV_VID);
  sprintf(text,"TOW REPORT");
  write_header(startrow,startcol,endcol,text,REV_VID);

  start = (endcol-startcol-strlen(menu[0]))/2 + startcol;
  choice=popup(menu,"123",3,startrow+6,start-2,NONE,REV_VID,0);

  if (choice==0)         /* send report to printer */
    prnt=1;
  else if (choice==1)    /* send report to file only */
    prnt=0;
  else if (choice==2 || choice<0)  /* QUIT or escape key pressed */
  {
    cursor_off();
    get_options(4);
    return;
  }

  /* open input and output files */
  in=fopen("report.in","r");
  out=fopen("report.out","w");

  /* check for null pointer */
  if (!out)
  {
    sprintf(text1,"Can't open file REPORT.OUT!");
    sprintf(text2,"==> exit, see if hard disk is full <==");
    display_error(ERROR,text1,text2);
    fclose(in);
    cursor_off();
    get_options(4);
    return;
  }

  /* print status message */
```

```
sprintf(text,". . . Writing Report . . .");
start = (endcol-startcol-strlen(text))/2 + startcol;
write_string(startrow+8,start,text,REV_VID);
sprintf(text,"Please Wait");
start = (endcol-startcol-strlen(text))/2 + startcol;
write_string(startrow+10,start,text,BLINK_REV_VID);

/* read messages */
for (i=0; i<9; i++)
{
  fgets(mssg[i],81,in);
  stripcr(mssg[i]);
}

/* print report header */
for (i=0; i<3; i++)
{
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
}
fprintf(out,"\n");
fprintf(out,"\n");

/*******************************************/
/*            print Section I              */
/*******************************************/

/* print header */
for (i=0; i<2; i++)
{
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
}
fprintf(out,"\n");
fprintf(out,"\n");

/* print tug class */
fgets(inline,81,in);
stripcr(inline);
pad(inline,1);
strcat(inline,tug);
fprintf(out,"%s\n",inline);
fprintf(out,"\n");

/* print hawser data */
fgets(inline,81,in);
fprintf(out,"%s",inline);              /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,3);
sprintf(text,"%6.2f in",tug_data[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);            /* diameter */
```

```
fgets(inline,81,in);
stripcr(inline);
pad(inline,6);
sprintf(text,"%6.0f ft",tug_data[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);           /* scope */
fprintf(out,"\n");

/* print chain pendant data */
fgets(inline,81,in);
fprintf(out,"%s",inline);             /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,7);
sprintf(text,"%6.2f in",tug_data[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);           /* size */

fgets(inline,81,in);
stripcr(inline);
pad(inline,6);
sprintf(text,"%6.0f ft",tug_data[4]);
strcat(inline,text);
fprintf(out,"%s\n",inline);           /* scope */
fprintf(out,"\n");
fprintf(out,"\n");

/*******************************************/
/*            print Section II             */
/*******************************************/
                /********/
if (type==0)    /* SHIP */
{               /********/
  /* print header */
  for (i=0; i<2; i++)
  {
    fgets(inline,81,in);
    fprintf(out,"%s",inline);
  }
  fprintf(out,"\n");
  fprintf(out,"\n");

  /* print tow type */
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
  fprintf(out,"\n");

  /* print input data */
  fgets(inline,81,in);
  fprintf(out,"%s",inline);           /* header */

  fgets(inline,81,in);
```

```
stripcr(inline);
pad(inline,1);
strcat(inline,hull_no);
fprintf(out,"%s\n",inline);          /* hull number */

fgets(inline,81,in);
stripcr(inline);
pad(inline,11);
sprintf(text,"%6.0f tons",tow_data[0]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* full load displacement */

fgets(inline,81,in);
stripcr(inline);
pad(inline,23);
sprintf(text,"%6.1f kts",tow_data[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tow speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,5);
sprintf(text,"%6.1f kts",tow_data[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* max wind speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,9);
sprintf(text,"%6.1f deg",tow_data[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* rel wind direction */

fgets(inline,81,in);
stripcr(inline);
pad(inline,14);
if (tow_data[4]==0.0)
  sprintf(text,"Locked");
else if (tow_data[4]==1.0)
  sprintf(text,"Trailing");
else if (tow_data[4]==2.0)
  sprintf(text,"Removed");
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* propeller status */
fprintf(out,"\n");

/* print Table G-2 data */
fgets(inline,81,in);
fprintf(out,"%s",inline);            /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,1);
strcat(inline,class);
```

```
  fprintf(out,"%s\n",inline);          /* ship class */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,10);
  sprintf(text,"%6.0f tons",ship_data[0]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* displacement */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,20);
  sprintf(text,"%6.0f sq ft",ship_data[1]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* frontal area */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,16);
  sprintf(text,"%6.2f",ship_data[2]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* wind coefficient */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,18);
  sprintf(text,"%6.0f sq ft",ship_data[3]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* propeller area */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,11);
  sprintf(text,"%6.0f",ship_data[4]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* hull resistance curve */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,11);
  sprintf(text,"%6.0f",ship_data[5]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* wave resistance curve */
  fprintf(out,"\n");
  fprintf(out,"\n");

  /* read and discard remaining Section II data */
  for (i=0; i<46; i++)
    fgets(inline,81,in);
}
                      /***********/
else if (type==1)     /* DRYDOCK */
{                     /***********/
  /* read and discard Section II data for type SHIP */
```

```
for (i=0; i<18; i++)
  fgets(inline,81,in);

/* print header */
for (i=0; i<2; i++)
{
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
}
fprintf(out,"\n");
fprintf(out,"\n");

/* print tow type */
fgets(inline,81,in);
fprintf(out,"%s",inline);
fprintf(out,"\n");

/* print input data */
fgets(inline,81,in);
fprintf(out,"%s",inline);           /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,1);
strcat(inline,hull_no);
fprintf(out,"%s\n",inline);         /* hull number */

fgets(inline,81,in);
stripcr(inline);
pad(inline,18);
sprintf(text,"%6.1f",tow_data[4]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* hull condition */

fgets(inline,81,in);
stripcr(inline);
pad(inline,23);
sprintf(text,"%6.1f kts",tow_data[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* tow speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,5);
sprintf(text,"%6.1f kts",tow_data[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* wind speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,9);
sprintf(text,"%6.1f deg",tow_data[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* wind direction */
```

```
fprintf(out,"\n");

/* print estimated displacement */
fgets(inline,81,in);
stripcr(inline);
pad(inline,13);
sprintf(text,"%6.0f tons",tow_data[0]);
strcat(inline,text);
fprintf(out,"%s\n",inline);
fprintf(out,"\n");

/* print Table G-4 data */
fgets(inline,81,in);
fprintf(out,"%s",inline);           /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,30);
sprintf(text,"%6.2f",dock_data[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* f1 */

fgets(inline,81,in);
stripcr(inline);
pad(inline,30);
sprintf(text,"%6.2f",dock_data[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* f2 */

fgets(inline,81,in);
stripcr(inline);
pad(inline,30);
sprintf(text,"%6.2f",dock_data[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* f3 */

fgets(inline,81,in);
stripcr(inline);
pad(inline,13);
sprintf(text,"%6.0f sq ft",dock_data[4]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* wetted surface area */

/* print cross sectional areas */
fgets(inline,81,in);
fprintf(out,"%s",inline);           /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,15);
sprintf(text,"%6.0f sq ft",dock_data[5]);
strcat(inline,text);
fprintf(out,"%s\n",inline);         /* below waterline */
```

```
  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,15);
  sprintf(text,"%6.0f",dock_data[6]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* above waterline */
  fprintf(out,"\n");
  fprintf(out,"\n");

  /* read and discard remaining Section II data */
  for (i=0; i<28; i++)
    fgets(inline,81,in);
}
                      /**********/
else if (type==2)     /* BARGE */
{                     /**********/
  /* read and discard Section II data for type SHIP, DRYDOCK */
  for (i=0; i<36; i++)
    fgets(inline,81,in);

  /* print header */
  for (i=0; i<2; i++)
  {
    fgets(inline,81,in);
    fprintf(out,"%s",inline);
  }
  fprintf(out,"\n");
  fprintf(out,"\n");

  /* print tow type */
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
  fprintf(out,"\n");

  /* print input data */
  fgets(inline,81,in);
  fprintf(out,"%s",inline);            /* header */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,1);
  strcat(inline,hull_no);
  fprintf(out,"%s\n",inline);          /* hull number */

  /* print hull dimensions */
  fgets(inline,81,in);
  fprintf(out,"%s",inline);            /* header */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,24);
  sprintf(text,"%6.1f ft",barge_data[0]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* length */
```

```
fgets(inline,81,in);
stripcr(inline);
pad(inline,26);
sprintf(text,"%6.1f ft",barge_data[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* beam */

fgets(inline,81,in);
stripcr(inline);
pad(inline,25);
sprintf(text,"%6.1f ft",barge_data[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* depth */

fgets(inline,81,in);
stripcr(inline);
pad(inline,25);
sprintf(text,"%6.1f ft",barge_data[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* draft */

/* print deckhouse dimensions */
fgets(inline,81,in);
fprintf(out,"%s",inline);            /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,24);
sprintf(text,"%6.1f ft",barge_data[4]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* length */

fgets(inline,81,in);
stripcr(inline);
pad(inline,25);
sprintf(text,"%6.1f ft",barge_data[5]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* width */

fgets(inline,81,in);
stripcr(inline);
pad(inline,24);
sprintf(text,"%6.1f ft",barge_data[6]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* height */

fgets(inline,81,in);
stripcr(inline);
pad(inline,16);
if (barge_data[7]==0.0)
  sprintf(text,"Rake ended");
else if (barge_data[7]==1.0)
  sprintf(text,"Ship ended");
```

```
else if (barge_data[7]==2.0)
  sprintf(text,"Square ended");
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* end shape */

fgets(inline,81,in);
stripcr(inline);
pad(inline,18);
sprintf(text,"%6.1f",tow_data[4]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* hull condition */

fgets(inline,81,in);
stripcr(inline);
pad(inline,23);
sprintf(text,"%6.1f kts",tow_data[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tow speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,5);
sprintf(text,"%6.1f kts",tow_data[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* wind speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,9);
sprintf(text,"%6.1f deg",tow_data[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* wind direction */
fprintf(out,"\n");

/* print estimated displacement */
fgets(inline,81,in);
stripcr(inline);
pad(inline,13);
sprintf(text,"%6.0f tons",tow_data[0]);
strcat(inline,text);
fprintf(out,"%s\n",inline);
fprintf(out,"\n");

/* print data corresponding to Table G-4 */
fgets(inline,81,in);
fprintf(out,"%s",inline);            /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,30);
sprintf(text,"%6.2f",barge_res[0]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* f1 */
```

```
  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,30);
  sprintf(text,"%6.2f",barge_res[1]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* f2 */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,30);
  sprintf(text,"%6.2f",barge_res[2]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* f3 */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,13);
  sprintf(text,"%6.0f sq ft",barge_res[3]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* wetted surface area */

  /* print cross sectional areas */
  fgets(inline,81,in);
  fprintf(out,"%s",inline);            /* header */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,15);
  sprintf(text,"%6.0f sq ft",barge_res[4]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* below waterline */

  fgets(inline,81,in);
  stripcr(inline);
  pad(inline,15);
  sprintf(text,"%6.0f",barge_res[5]);
  strcat(inline,text);
  fprintf(out,"%s\n",inline);          /* above waterline */
  fprintf(out,"\n");
  fprintf(out,"\n");
}
fprintf(out,"\f");                     /* form feed */

/*********************************************/
/*            print Section III              */
/*********************************************/

/* print header */
for (i=0; i<2; i++)
{
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
}
fprintf(out,"\n");
```

```
fprintf(out,"\n");

/* print tow type */
fgets(inline,81,in);
stripcr(inline);
pad(inline,7);
if (type==0)
  sprintf(text,"SHIP");
else if (type==1)
  sprintf(text,"DRYDOCK");
else if (type==2)
  sprintf(text,"BARGE");
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tow type */
fprintf(out,"\n");

/* print data */
fgets(inline,81,in);
stripcr(inline);
pad(inline,13);
sprintf(text,"%8.0f lbs",resist_dat[0]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* wind resistance */

fgets(inline,81,in);
stripcr(inline);
pad(inline,7);
sprintf(text,"%6.1f ft",resist_dat[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* wave height */

fgets(inline,81,in);
stripcr(inline);
pad(inline,13);
sprintf(text,"%8.0f lbs",resist_dat[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* wave resistance */

fgets(inline,81,in);
stripcr(inline);
pad(inline,13);
sprintf(text,"%8.0f lbs",resist_dat[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* hull resistance */

fgets(inline,81,in);
stripcr(inline);
pad(inline,8);
sprintf(text,"%8.0f lbs",resist_dat[4]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* propeller resistance */

fgets(inline,81,in);
fprintf(out,"%s",inline);
```

```
tow_res=resist_dat[0];
for (i=2; i<5; i++)
  tow_res += resist_dat[i];

fgets(inline,81,in);
stripcr(inline);
pad(inline,8);
sprintf(text,"%8.0f lbs",tow_res);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* total tow resistance */
fprintf(out,"\n");

hawser_resist(tug_data,tow_data[1],tow_res,&haw_res);
fgets(inline,81,in);
stripcr(inline);
pad(inline,16);
sprintf(text,"%8.0f lbs",haw_res);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* hawser resistance */
fprintf(out,"\n");

fgets(inline,81,in);
stripcr(inline);
pad(inline,13);
sprintf(text,"%8.0f lbs",tow_res+haw_res);
strcat(inline,text);
sprintf(text,"  or %6.2f kips",(tow_res+haw_res)/1000.0);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* mean towline tension */
fprintf(out,"\n");
fprintf(out,"\n");

/*******************************************/
/*            print Section IV             */
/*******************************************/

/* print header */
for (i=0; i<2; i++)
{
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
}
fprintf(out,"\n");
fprintf(out,"\n");

/* print tug, tow */
fgets(inline,81,in);
stripcr(inline);
pad(inline,1);
strcat(inline,tug);
fprintf(out,"%s\n",inline);          /* tug */
fgets(inline,81,in);
stripcr(inline);
```

```
pad(inline,1);
strcat(inline,hull_no);
fprintf(out,"%s\n",inline);          /* tow */
fprintf(out,"\n");

/* print data */
fgets(inline,81,in);
stripcr(inline);
pad(inline,14);
sprintf(text,"%8.1f kts",tug_eval[0]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* desired tow speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,5);
sprintf(text,"%8.0f lbs",tug_eval[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* mean tension */

fgets(inline,81,in);
stripcr(inline);
sprintf(text,"%8.0f lbs",tug_eval[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* available tension */
fprintf(out,"\n");

/* print message */
if (tug_eval[2]>=tug_eval[1])         /* tug has sufficient pull */
  fprintf(out,"%s\n",mssg[1]);
else if (tug_eval[2]<tug_eval[1])     /* tug has insufficient pull */
{
  fprintf(out,"%s\n",mssg[0]);
  fprintf(out,"%s\n",mssg[2]);
}
fprintf(out,"\n");

/* print remaining data */
fgets(inline,81,in);
stripcr(inline);
pad(inline,8);
sprintf(text,"%8.1f kts",tug_eval[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* best possible tow speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,5);
sprintf(text,"%8.0f lbs",tug_eval[4]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* mean tension */

fgets(inline,81,in);
stripcr(inline);
```

```
sprintf(text,"%8.0f lbs",tug_eval[5]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* available tension */
fprintf(out,"\n");

fgets(inline,81,in);
fprintf(out,"%s",inline);

fgets(inline,81,in);
stripcr(inline);
if (tug_eval[6]==1.0)
  sprintf(text,"%5.1f kts (ORIGINAL)",tug_eval[0]);
if (tug_eval[6]==0.0)
  sprintf(text,"%5.1f kts (BEST POSSIBLE)",tug_eval[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* selected tow speed */
fprintf(out,"\n");
fprintf(out,"\f");                   /* form feed */

/*******************************************/
/*            print Section V              */
/*******************************************/

/* print header */
for (i=0; i<2; i++)
{
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
}
fprintf(out,"\n");
fprintf(out,"\n");

/* print tug, tow */
fgets(inline,81,in);
stripcr(inline);
pad(inline,1);
strcat(inline,tug);
fprintf(out,"%s\n",inline);          /* tug */
fgets(inline,81,in);
stripcr(inline);
pad(inline,1);
strcat(inline,hull_no);
fprintf(out,"%s\n",inline);          /* tow */
fprintf(out,"\n");

/* print tabulated data */
fgets(inline,81,in);
fprintf(out,"%s",inline);            /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,1);
i=(int)ext_data[0];
strcat(inline,tabtow[i]);
```

```
fprintf(out,"%s\n",inline);          /* tabulated tow */

fgets(inline,81,in);
stripcr(inline);
pad(inline,15);
sprintf(text,"%6.0f tons",ext_data[1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tabulated displacement */

fgets(inline,81,in);
stripcr(inline);
pad(inline,18);
sprintf(text,"%6.0f kts",ext_data[2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tabulated tow speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,17);
sprintf(text,"%6.0f kts",ext_data[3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tabulated wind speed */

fgets(inline,81,in);
stripcr(inline);
pad(inline,13);
sprintf(text,"%6.0f deg",ext_data[4]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tabulated wind direction */

fgets(inline,81,in);
stripcr(inline);
pad(inline,15);
sprintf(text,"%6.0f ft",ext_data[5]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tabulated hawser scope */

fgets(inline,81,in);
fprintf(out,"%s",inline);
fgets(inline,81,in);
stripcr(inline);
pad(inline,10);
sprintf(text,"%6.0f",ext_data[6]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* curve number */
fprintf(out,"\n");

/* print results */
fgets(inline,81,in);
fprintf(out,"%s",inline);            /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,15);
```

```
if (ext_data[9]==0)
  sprintf(text,"%6.2f kips (ESTIMATED)",ext_data[7]);
else if (ext_data[9]==1)
  sprintf(text,"%6.2f kips (ACTUAL)",ext_data[7]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* mean tension */

fgets(inline,81,in);
stripcr(inline);
pad(inline,12);
sprintf(text,"%6.2f kips",ext_data[8]-ext_data[7]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* dynamic tension */

fgets(inline,81,in);
stripcr(inline);
pad(inline,12);
sprintf(text,"%6.2f kips",ext_data[8]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* extreme tension */
fprintf(out,"\n");

/* print warnings, if applicable */
if (ext_data[7]>100.0)
{
  fprintf(out,"%s",mssg[0]);
  fprintf(out,"%s",mssg[3]);
  fprintf(out,"%s",mssg[5]);
  fprintf(out,"%s",mssg[6]);
  fprintf(out,"\n");
}
if (ext_data[8]>400.0)
{
  fprintf(out,"%s",mssg[0]);
  fprintf(out,"%s",mssg[4]);
  fprintf(out,"%s",mssg[5]);
  fprintf(out,"%s",mssg[6]);
  fprintf(out,"\n");
}

/* print note regarding basis for speed,length variations */
for (i=0; i<2; i++)
{
  fgets(inline,81,in);
  fprintf(out,"%s",inline);
}
fprintf(out,"\n");

/* print speed variation data */
fgets(inline,81,in);
fprintf(out,"%s",inline);            /* header */

fgets(inline,81,in);
stripcr(inline);
```

```
pad(inline,7);
sprintf(text,"%10.1f%10.1f%10.1f",spd_data[0][0],spd_data[1][0],
              spd_data[2][0]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tow speed */
fprintf(out,"\n");

fgets(inline,81,in);
stripcr(inline);
pad(inline,10);
sprintf(text,"%10.0f%10.0f%10.0f",spd_data[0][1],spd_data[1][1],
              spd_data[2][1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* curve number */
fprintf(out,"\n");

fgets(inline,81,in);
stripcr(inline);
pad(inline,3);
sprintf(text,"%10.1f%10.1f%10.1f",spd_data[0][2],spd_data[1][2],
              spd_data[2][2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* mean tension */

fgets(inline,81,in);
stripcr(inline);
x1=spd_data[0][3]-spd_data[0][2];
x2=spd_data[1][3]-spd_data[1][2];
x3=spd_data[2][3]-spd_data[2][2];
sprintf(text,"%10.1f%10.1f%10.1f",x1,x2,x3);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* dynamic tension */

fgets(inline,81,in);
stripcr(inline);
sprintf(text,"%10.1f%10.1f%10.1f",spd_data[0][3],spd_data[1][3],
              spd_data[2][3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* extreme tension */
fprintf(out,"\n");
fprintf(out,"\n");

/* print length variation data */
fgets(inline,81,in);
fprintf(out,"%s",inline);            /* header */

fgets(inline,81,in);
stripcr(inline);
pad(inline,5);
sprintf(text,"%10.0f%10.0f%10.0f",lgth_data[0][0],lgth_data[1][0],
              lgth_data[2][0]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* tow speed */
fprintf(out,"\n");
```

```
fgets(inline,81,in);
stripcr(inline);
pad(inline,10);
sprintf(text,"%10.0f%10.0f%10.0f",lgth_data[0][1],lgth_data[1][1],
            lgth_data[2][1]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* curve number */
fprintf(out,"\n");

fgets(inline,81,in);
stripcr(inline);
pad(inline,3);
sprintf(text,"%10.1f%10.1f%10.1f",lgth_data[0][2],lgth_data[1][2],
            lgth_data[2][2]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* mean tension */

fgets(inline,81,in);
stripcr(inline);
x1=lgth_data[0][3]-lgth_data[0][2];
x2=lgth_data[1][3]-lgth_data[1][2];
x3=lgth_data[2][3]-lgth_data[2][2];
sprintf(text,"%10.1f%10.1f%10.1f",x1,x2,x3);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* dynamic tension */

fgets(inline,81,in);
stripcr(inline);
sprintf(text,"%10.1f%10.1f%10.1f",lgth_data[0][3],lgth_data[1][3],
            lgth_data[2][3]);
strcat(inline,text);
fprintf(out,"%s\n",inline);          /* extreme tension */
fprintf(out,"\f");                   /* form feed */

fclose(in);
fclose(out);


/* send report file to printer */
if (prnt)
{
  /* prompt user to check printer before proceeding */
  sprintf(text1,"Ensure printer is on line");
  sprintf(text2,"and ready to go");
  display_error(WARN,text1,text2);
  cursor_off();

  /* print status message */
  sprintf(text,". . . Printing Report . . .");
  start = (endcol-startcol-strlen(text))/2 + startcol;
  write_string(startrow+8,start,text,REV_VID);
  sprintf(text,"Please Wait");
  start = (endcol-startcol-strlen(text))/2 + startcol;
```

```
    write_string(startrow+10,start,text,BLINK_REV_VID);

    /* print report */
    print_rep();
  }
  clear(startrow+1,startcol+1,endrow-1,endcol-1);
  return;
}

print_rep()
{
  FILE *in;
  char str[81];

  in=fopen("report.out","r");

  while ( !feof(in) )
  {
    if (fgets(str,81,in))
      prints(str);
  }
}

prints(s)
char *s;
{
  while (*s) bdos(0x5,*s++,0);
}
```

```
/*************************************************************************
  File: plt.c
  Author: J. H. Milgram
  Edited: Todd J. Peltzer
  Last update: 30 April 1989

    This file contains plotting functions written for the ATHENA lab
  computer, using Halo 88 functions. Modified to work with TOWCALC.
*************************************************************************/
#include "stdio.h"
#include "plthead.h"
#include "string.h"

setplt()
{
  FILE *fp;
  int status,nbr,n;
  char ch1, ch2, gdevice[13],inline[81];

  fp = fopen("device.dat","r");
  n=fscanf(fp,"%s\n",gdevice);        /* read device driver from file */
  n=fscanf(fp,"%d\n",&mode);          /* read graphics mode from file */

  fclose(fp);
  setdev( gdevice );
  scxx = 8.28;
  scyy = 5.52;
  cnul = '\0';
  knumx = 0;
  knumy = 0;
  x0 = 0.13;
  y0p = 0.10;
  xl = 0.80;
  yl = 0.87;
}

locate(x0i,y0i,xli,yli)
float *x0i,*y0i,*xli,*yli;
{
  x0 = *x0i / scxx;
  y0p = *y0i / scyy;
  xl = *xli / scxx;
  yl = *yli / scyy;
}

line3(npts,x,y)
short *npts;
float x[], y[] ;
{
  short npt;
  float xm1,ym1;
  xm1 = x[0];
  ym1 = y[0];
  movabs( &xm1, &ym1);
```

```
  npt = *npts - 1;
  polylnabs( x, y, &npt );
)

xyaxis(xmn,xmx,xtc,ymn,ymx,ytc,xf,yf,kod)
float *xmn,*xmx,*xtc,*ymn,*ymx,*ytc,*xf,*yf;
short *kod;
{
  float xdel,ydel,xsmall,ysmall,xbig,ybig,xmnp,ymnp,xmxp,ymxp,zero;
  short ione, izr, kode, i, mn, mm, m1, m2;
  short format = 1, height = 1, page = 1, screen = 1, width = 1;

  ione = 1;
  izr = 0;
  xdel = *xmx - *xmn;
  ydel = *ymx - *ymn;
  xsmall = *xmn - (x0 / xl) * xdel;
  xbig = *xmx + ((1.0 - x0 - xl) / xl) * xdel;
  ysmall = *ymn - (y0p / yl) * ydel;
  ybig = *ymx + ((1.0 - y0p - yl) / yl) * ydel;
  xrange = xdel / xl;
  yrange = ydel / yl;
  initgraphics( &mode );
  setieee( &format );
  display( &page );
  setscreen( &screen );
  setworld( &xsmall, &ysmall, &xbig, &ybig );
  inqdrange( &npx, &npy );
  nfax = npx / scxx;
  nfay = npy / scyy;
  xmnp = *xmn;
  xmxp = *xmx;
  ymnp = *ymn;
  ymxp = *ymx;
  mapwtod( &xmnp, &ymnp, &ix0, &iy0);
  mapwtod( &xmxp, &ymxp, &ixl, &iyl);
  ptabs( &xmnp, &ymnp );
  inqclr( &xmnp, &ymnp, &icf );
  inqbknd( &icb );
  setcolor( &icb );
  ptabs( &xmnp, &ymnp );
  setcolor( &icf );
  inittcur( &height, &width, &icb );
  tclx = 0.011 * (*xmx - *xmn);
  tcly = 0.015 * (*ymx - *ymn);
  kode = *kod;
  xtic = *xtc;
  ytic = *ytc;
  if (kode >= 10)
    goto case1;
  movabs( &xmnp, &ymxp );
  lnabs( &xmnp, &ymnp );
  lnabs( &xmxp, &ymnp );
  if (kode == 1)
```

```
  goto case1;
lnabs( &xmxp, &ymxp );
lnabs( &xmnp, &ymxp );
case1: nxx = (1.0001 * (*xmx - *xf)) / xtic + 1;
nyy = (1.0001 * (*ymx - *yf)) / ytic + 1;
if (kode >= 10)
  kode = kode - 10;
xmin = *xmn;
xmax = *xmx;
ymin = *ymn;
ymax = *ymx;
xff = *xf;
yff = *yf;
for ( i = 1; i <= nxx; i++ )
  xtx[i] = (*xf + (i - 1) * xtic);
for ( i = 1; i <= nyy; i++)
  yty[i] = (*yf + (i - 1) * ytic);
if ( kode == 3 )
  goto case2;
for ( i = 1; i <= nxx; i++ )
{
  xtl[i] = *ymn - tcly;
  if ((kode == 2) || (kode == 4))  goto case01;
  xtu[i] = *ymn + tcly;
  goto case02;

  case01:  xtu[i] = *ymn;
  case02: ;
}
for ( i = 1; i <= nyy; i++ )
{
  ytl[i] = *xmn - tclx;
  if ((kode == 2) || (kode == 4))
    ytu[i] = *xmn;
  else
    ytu[i] = *xmn + tclx;
}
goto case3;
case2:  for ( i = 1; i <= nxx; i++ )
{
  xtl[i] = *ymn;
  xtu[i] = *ymx;
}
for ( i = 1; i <= nyy; i++ )
{
  ytl[i] = *xmn;
  ytu[i] = *xmx;
}
case3:
  if (*yf >= ymax)  goto case5;
mm = nyy / 2 + 1;
for ( i = 1; i <= mm; i++)
{
  m1 = nyy - 2 * (i -1);
```

```
  if (m1 < 1)  goto case4;
  drawu( &ytl[m1], &yty[m1], &ytu[m1], &yty[m1] );
  m2 = m1 - 1;
  if ( m2 < 1)  goto case4;
  drawu( &ytu[m2], &yty[m2], &ytl[m2], &yty[m2] );
  case4: ;
}
case5:  if (*xf > xmax)  goto case7;

mn = nxx / 2 + 1;
for ( i = 1; i <= mn; i++ )
{
  m1 = 1 + 2 * (i - 1};
  if (m1 > nxx)  goto case6;
  drawu( &xtx[m1], &xtl[m1], &xtx[m1], &xtu[m1] );
  m2 = m1 + 1;
  if( m2 > nxx)  goto case6;
  drawu( &xtx[m2], &xtu[m2], &xtx[m2], &xtl[m2]);
  case6: ;
}
case7: if ((kode == 1) || (kode == 3))  goto case12;
for ( i = 1; i <= nyy; i++)
{
  y3l[i] = *xmx - tclx;
  y3u[i] = *xmx;
}
for ( i = 1; i <= nxx; i++ )
{
  x3l[i] = *ymx - tcly;
  x3u[i] = *ymx;
}
if (*yf >= ymax}  goto case9;
for ( i = 1; i <= mm; i++)
{
  m1 = 1 + 2 * (i - 1);
  if (m1 > nyy)  goto case8;
  drawu(  &y3l[m1], &yty[m1], &y3u[m1], &yty[m1] );
  m2 = m1 + 1;
  if (m2 > nyy) goto case8;
  drawu( &y3u[m2], &yty[m2], &y3l[m2], &yty[m2] );
  case8:  ;
}
case9:  if (*xf >= xmax)  goto case11;
for ( i = 1; i <= mn; i++ )
{
  m1 = nxx - 2 * (i-1);
  if (m1 < 1)  goto case10;
  drawu( &xtx[m1], &x3l[m1], &xtx[m1], &x3u[m1] );
  m2 = m1 - 1;
  if (m2 < 1)  goto case10;
  drawu( &xtx[m2], &x3u[m2], &xtx[m2], &x3l[m2] );
  case10:  ;
}
case11:  if (kode != 4}  goto case12;
```

```
  zero = 0.0;
  drawu( &zero, &ymnp, &zero, &ymxp);
  drawu( &xmnp, &zero, &xmxp, &zero);
  case12:  ;
}

drawu(x1,y1,x2,y2)
float *x1,*y1,*x2,*y2;
{
  float xx1,yy1,xx2,yy2;
  xx1 = *x1;
  yy1 = *y1;
  xx2 = *x2;
  yy2 = *y2;
  movabs(&xx1,&yy1);
  lnabs(&xx2,&yy2);
}

label(itype,string)
int *itype;
char  string[81]  ;
{
  char cdl,straux[81],cnull=NULL,*result;
  int h = 1, w = 1, hor = 0, ver = 1, str = 0;
  int n,nm ;
  float yw1,xlaby,xxn,yyn,xn1,yn1,xx,yy;
  cdl = '_'  ;
  strset(straux,cnull);
  if ( *itype != 1  &&  *itype != 2 )  goto casend;
  n = strlen(string);
  nm = n-1;
  if (string[nm] == cdl)
  {
    strncpy(straux,string,nm);
    straux[nm] = cnull;
    n = nm  ;
  }
  if (string[nm] != cdl)
  {
    strncpy(straux,string,n);
    straux[n] = cnull ;
  }
  if (*itype == 1)
  {
    xxn = x0 + 0.5 * ( xl - 0.01 * (n + 2) );
    yyn = 1.0 - y0p + 0.1;
    settext(&h,&w,&hor,&str);
  }
  else   goto case3 ;
  goto case4 ;
  case3:  xlaby = xmin - 0.070 * xrange ;
  yw1 = 0.0 ;
  mapwton( &xlaby, &yw1, &xn1,&yn1 ) ;
  xxn = xn1 - 2. * 0.018 ;
```

```
  yyn = 1.0 - y0p + 0.5 * ( 0.023 * n - yl );
  settext(&h, &w, &ver, &str);
  case4:   mapntow(&xxn, &yyn, &xx, &yy);
  movtcurabs(&xx,&yy);
  btext (straux) ;
  casend: ;
}

labely(x,y,string)
float *x, *y ;
char  string[81]  ;
{
  char cdl,straux[81],cnull=NULL,*result;
  int h = 1, w = 1, hor = 0, ver = 1, str = 0;
  int n,nm ;
  float xl,yl;
  cdl = '_';
  strset(straux,cnull);
  n = strlen(string);
  nm = n-1;
  if (string[nm] == cdl)
  {
    strncpy(straux,string,nm);
    straux[nm] = cnull;
  }
  if (string[nm] != cdl)
  {
    strncpy(straux,string,n);
    straux[n] = cnull ;
  }
  settext(&h,&w,&hor,&str);
  xl = *x;
  yl = *y;
  movtcurabs(&xl,&yl);
  btext( straux );
}

endplt()
{
  closegraphics();
}

symplt(npts,x,y,nsym)
int *npts,*nsym;
float x[], y[];
{
  float dx,dy,xx,yy,xxl,xxm,xxr,yyb,yym,yyt,dd,yyy,ddx;
  int sty = 1 , i,nsymb,ncode,npnt;
  sethatchstyle(&sty);
  ncode = *nsym / 10;
  nsymb = *nsym -  10 * ncode;
  dx = 0.006 * xrange;
  dy = 0.008 * yrange;
  npnt = *npts - 1;
```

```
for ( i = 0 ; i <= npnt ; ++i )
{
  xx = x[i];
  yy = y[i];
  xxl = xx - dx;
  xxm = xx;
  xxr = xx+ dx;
  yyb = yy - dy;
  yym = yy;
  yyt = yy + dy;
  if (nsymb != 0) goto case1;
  movabs(&xxl,&yym);
  lnabs (&xxr,&yym);
  movabs(&xxm,&yyt);
  lnabs(&xxm,&yyb);
  case1:   if (nsymb != 1)  goto case2;
  movabs(&xxl,&yyb);
  lnabs(&xxr,&yyt);
  movabs(&xxl,&yyt);
  lnabs (&xxr,&yyb);
  case2:   if (nsymb != 2)  goto case3;
  movabs(&xxl,&yyb);
  lnabs (&xxr,&yyb);
  lnabs (&xxr,&yyt);
  lnabs (&xxl,&yyt);
  lnabs (&xxl,&yyb);
  if (ncode == 0) goto case3;
  movabs(&xx,&yy);
  flood(&icf);
  case3:   if (nsymb != 3)  goto case4;
  movabs(&xxl,&yym);
  lnabs (&xxm,&yyb);
  lnabs (&xxr,&yym);
  lnabs (&xxm,&yyt);
  lnabs (&xxl,&yym);
  if (ncode == 0)  goto case4 ;
  movabs(&xx,&yy);
  flood(&icf);
  case4:   if (nsymb != 4) goto case5;
  movabs(&xxm,&yyt);
  lnabs(&xxl,&yyb);
  lnabs(&xxr,&yyb);
  lnabs (&xxm,&yyt);
  if (ncode == 0) goto case5;
  movabs(&xx,&yy);
  flood(&icf);
  case5:   if (nsymb != 5) goto case6;
  movabs(&xxr,&yyt);
  lnabs(&xxl,&yyb);
  lnabs (&xxr,&yyb);
  lnabs (&xxl,&yyt);
  lnabs (&xxr,&yyt);
  if (ncode == 0) goto case6;
  dd = 0.005 * yrange;
```

```
    yyy = yy + dd;
    movabs (&xx,&yyy);
    flood(&icf);
    yyy = yy - dd;
    movabs(&xx,&yyy);
    flood(&icf);
    case6:   if (nsymb != 6) goto case7;
    movabs(&xx,&yy);
    ddx = 2.0 * dx;
    cir(&ddx);
    if (ncode == 0) goto case7;
    movabs(&xx,&yy);
    flood(&icf);
    case7:   ptabs(&xx,&yy);
  )
)

eraseu(x1,y1,x2,y2)
float *x1, *y1, *x2, *y2;
{
  float xo,yo,xt,yt;
  xo = *x1;
  yo = *y1;
  xt = *x2;
  yt = *y2;
  setcolor( &icb );
  movabs( &xo, &yo );
  lnabs( &xt, &yt );
  setcolor( &icf );
)

axnum(ntx,nty)
short *ntx , *nty ;
{
  static char *cmft[] =
  {"    ", "%5.0f", "%5.1f", "%5.2f", "%5.3f",
                          "%5.4f", "%5d", "%  3d"
  );

  static char *clft[] =
  {"    ", "%6.0f", "%6.1f", "%6.2f", "%6.3f",
                          "%6.4f", "%5d", "%  3d"
  );

  short i, jx, jy, ival, ione = 1, izero = 0 ;
  float yy,xx , val ;
  char cvl[12] , cvs[12] ;

  ntix = *ntx  ;
  ntiy = *nty  ;
  jx = 7    ;
  jy = 7    ;
  settext(&ione,&ione,&izero,&izero) ;
  if ( knumx == 1 ) goto case1;
```

```
  jx = 1  ;
  if ( (xmax >  9999.5) || (xmin <  -999.95) )  jx = 6 ;
  if ( (xmax < 999.95) && (xmin > -99.95) )  jx = 2  ;
  if ( (xmax < 99.995) && (xmin > -9.995) )  jx = 3  ;
  if( (xmax < 9.9995) && (xmin > -0.9995) ) jx = 4   ;
  if( (xmax < 0.99995) && (xmin >= 0.000) )  jx = 5  ;

  case1: if ( knumy == 1 ) goto case2;
  jy = 1  ;
  if ( (ymax > 9999.5)  || (ymin < -999.95) )  jy = 6 ;
  if ( (ymax < 999.95) && (ymin > -99.95) )  jy = 2   ;
  if ( (ymax < 99.995) && (ymin > -9.995) )  jy = 3   ;
  if ( (ymax < 9.9995) && (ymin > -0.9995) ) jy = 4   ;
  if ( (ymax < 0.99995) && (ymin >= 0.00) )   jy = 5  ;
  case2: ;

  for (i = 1 ; i <= nxx ; i = i + ntix)
  {
    yy = ymin - 0.045 * yrange ;
    xx = xtx[i] - 0.045 * xrange ;
    val = xff + (i-1.) * xtic ;
    ival = val ;
    if (jx >=   6) sprintf (cvl, clft[jx],ival) ;
    if (jx  < 6) sprintf (cvl,clft[jx],val) ;
    movtcurabs(&xx,&yy) ;
    btext(cvl) ;
  }
  for (i = 1 ; i <= nyy ; i = i + ntiy )
  {
    xx= xmin - 0.070 * xrange ;
    yy= yty[i] - 0.013 * yrange ;
    val=yff + (i-1) * ytic ;
    ival = val ;
    if (jy  >= 6) sprintf(cvs,cmft[jy], ival) ;
    if (jy  <  6) sprintf(cvs,cmft[jy],val) ;
    movtcurabs(&xx,&yy)  ;
    btext(cvs) ;
  }
}

waitke()
{
  /* WAITING...*/
  /* LOOP UNTIL kbhit() REPORTS A KEYSTROKE...*/

  while( !kbhit() ) ;
)
```

```
/*************************************************************************
  File: tlib.c
  Author: Todd J. Peltzer
  Last update: 30 April 1989

     This file combines the files iolib1.c, popup.c, and video.c
*-----------------------------------------------------------------------*
   iolib1.c

   Author: Norbert H. Doerry
   Edited: Todd J. Peltzer
   Last update: 30 April 1989

     This file contains a set of routines to augment the IO functions
   in the standard IO libraries.

   Note: this version of iolib combines NHD's original iolib and ioliba
   files along with modifications and additions by TJP.

-------------------------------------------------------------------------
  Functions:
    stofa()       : "string to floating array"
    stoda()       : "string to double array"
    stoia()       : "string to short integer array"
    parse()       : parse a string into its elements
    suctolc()     : "string upper case to lower case"
    slctouc()     : "string lower case to upper case"
    strsplit()    : "string split"
    strstrip()    : "string strip"
    stripcr()     : "strip carriage return"
    bioskey()     : emulates part of the Turbo C bioskey() function
    get_key()     : reads the 16-bit scan code of a key
    get_special() : returns the position code of arrow and function keys
    is_in()       : tests if a character is in a given string
*************************************************************************/
#include "stdio.h"
#include "dos.h"
#include "stdlib.h"
#include "keydef.h"
#include "video.h"

int vmode;
char far *vid_mem;

/*************************************************************************
  stofa : "string to floating array"
  Norbert H. Doerry
  rev a 10 July 1988
  rev b 29 October 1988 (TJP):  added fourth argument; deleted flags.
  rev c 30 October 1988 (TJP):  added exponential notation.

  This function converts a string to an array of floating point numbers
  and passes back the array and the number of numbers successfully con-
  verted. The function returns zero if read successfully to the end of
```

```
  the line, otherwise it returns the character at which failure occurred.
  The fourth argument passed to the function is the maximum number of
  elements in the array.

   Note: stofa does not convert with 100% accuracy; roundoff error causes
         some loss of significance beyond 5 or 6 decimal places. Use the
         function "stoda()" if more significant digits are required.
*************************************************************************/
stofa(string,fltaptr,nbrptr,maxnum)
char string[];
float fltaptr[];
int *nbrptr,maxnum;
{
   int sign;
   int index = 0;
   int expsign, exponent;
   float power;
   char ch;
   float inch;
   *nbrptr = 0;

   while ((ch=string[index++]) == ' ' || ch == '\t') ;
        /* strip off leading blanks and tabs */

   while (((ch >= '0' && ch <= '9') || ch == '.' || ch == '+' || ch == '-'
           || ch == ',' || ch == ';' || ch == ':') && *nbrptr < maxnum)
   (
      sign = 1;
      power = 10;
      if ( ch == '-' || ch == '+')
         (
           if ( ch == '-')
              sign = -1;
           else
              sign = 1;
           ch = string[index++];
         }
      fltaptr[*nbrptr] = 0;   /* Initialize value */
      while (ch >= '0' && ch <= '9')
      (
        fltaptr[*nbrptr] = 10.0 * (fltaptr[*nbrptr]) + ch - '0';
        ch = string[index++];
      )
      if (ch == '.')    /* Check for decimal point */
         while ((ch = string[index++]) >= '0' && ch <= '9')
         {
           fltaptr[*nbrptr] = fltaptr[*nbrptr] + (ch - '0') / power;
           power *= 10;
         }
      /* Check for exponential notation */
      if (ch == 'E' || ch == 'e')
      (
         expsign = 1;
         if ( (ch = string[index++]) == '-' || ch == '+')
```

```
      {
        if ( ch == '-')
           expsign = -1;
        ch = string[index++];
    }
    exponent = 0;
    while (ch >= '0' && ch <= '9')
    {
      exponent = 10.0 * (exponent) + ch - '0';
      ch = string[index++];
    )
    power = (expsign == -1) ? 0.1 : 10;
    while (exponent-- >=1)
    {
       fltaptr[*nbrptr] *= power;
    }
}
else if (ch == '-') /* Check for "ft-in" entry */
{
    if ((ch = string[index++]) >= '0' && ch <='9')
    {
      inch = ch - '0';
      if (inch == 1 || inch == 0)
        if ((ch = string[index++]) == '0' || ch == '1')
          inch = 10*inch + ch - '0';
        else
          index--;
      power = 10;
      if ((ch = string[index++]) == '.')  /* ft-decimal inch */
      {
        while ((ch = string[index++]) >= '0' && ch <= '9')
        {
          inch = inch + (ch - '0') / power;
          power *= 10;
        }
      }
      else if (ch == '-')  /* ft-inch-eighth */
      {
        if ((ch = string[index++]) >= '0' && ch <= '7')
        {
          inch += (ch - '0')/(8.0);
          if ((ch = string[index++]) == '.')
            while ((ch = string[index++]) >= '0' && ch <= '9')
            {
              inch += (ch - '0')/(8.0*power);
              power *= 10;
            }
        )
      }
      fltaptr[*nbrptr] += inch / 12.0;
    }
}
fltaptr[*nbrptr] *= sign;
(*nbrptr)++;
```

```
      if (ch == '-' || ch == '+' || ch == '8' || ch == '9' || ch == '.')
        break;
      while (ch == ' ' || ch == '\t')
        ch = string[index++];
      if (ch == ':' || ch == ';' || ch == ',' )
         /* recognize these characters as delimiters */
        while ((ch = string[index++]) == ' ' || ch == '\t') ;
      while (ch == ':' || ch == ';' || ch == ',' )
      {
        if (*nbrptr < maxnum)
        {
          fltaptr[*nbrptr] = 0;
          (*nbrptr)++;
        }
        while ((ch = string[index++]) == ' ' || ch == '\t') ;
      }
   }
   return (ch);
}

/*************************************************************************
  stoda : "string to double array"
  Todd J. Peltzer
  after stofa by Norbert H. Doerry
  29 October 1988
  This function converts a string to an array of double precision numbers
  and passes back the array (via "dblaptr") and the number of numbers
  successfully converted (via "nbrptr"). The function returns zero if it
  read successfully to the end of the line, otherwise it returns the
  character at which failure occurred. The fourth argument passed to the
  function ("maxnum") is the maximum number of elements in the array.
  The function accepts numbers in either decimal or exponential form (but
  not ft-inch-eighth).
*************************************************************************/
stoda(string,dblaptr,nbrptr,maxnum)
char string[];
double dblaptr[];
int *nbrptr, maxnum;
{
   int sign;
   int expsign, exponent;
   int index = 0;
   double power;
   char ch;
   *nbrptr = 0;

   while ((ch=string[index++]) == ' ' || ch == '\t') ;
        /* . . . strip off leading blanks and tabs */

   while (((ch >= '0' && ch <= '9') || ch == '.' || ch == '+' || ch == '-'
           || ch == ',' || ch == ';' || ch == ':') && *nbrptr < maxnum )
        /* . . . test for allowable characters */
   {
      sign = 1;
```

```
power = 10;
if ( ch == '-' || ch == '+')
   {
     if ( ch == '-')
        sign = -1;
     ch = string[index++];
   }
dblaptr[*nbrptr] = 0;    /* Initialize value */
while (ch >= '0' && ch <= '9')
{
  dblaptr[*nbrptr] = 10.0 * (dblaptr[*nbrptr]) + ch - '0';
  ch = string[index++];
}
if (ch == '.')     /* Check for decimal point */
   while ((ch = string[index++]) >= '0' && ch <= '9')
   {
     dblaptr[*nbrptr] = dblaptr[*nbrptr] + (ch - '0') / power;
     power *= 10;
   }
/* Check for exponential notation */
if (ch == 'E' || ch == 'e')
{
   expsign = 1;
   if ( (ch = string[index++]) == '-' || ch == '+')
      {
        if ( ch == '-')
           expsign = -1;
        ch = string[index++];
   }
   exponent = 0;
   while (ch >= '0' && ch <= '9')
   {
     exponent = 10.0 * (exponent) + ch - '0';
     ch = string[index++];
   }
   power = (expsign == -1) ? 0.1 : 10;
   while (exponent-- >=1)
   {
      dblaptr[*nbrptr] *= power;
   }
}

dblaptr[*nbrptr] *= sign;
(*nbrptr)++;
if (ch == '-' || ch == '+' || ch == '.')
  break;
while (ch == ' ' || ch == '\t')
  ch = string[index++];
if (ch == ':' || ch == ';' || ch == ',' )
   /* recognize these characters as delimiters */
   while ((ch = string[index++]) == ' ' || ch == '\t') ;
while (ch == ':' || ch == ';' || ch == ',' )
{
  if (*nbrptr < maxnum)
```

```
        {
          dblaptr[*nbrptr] = 0;
          (*nbrptr)++;
        }
        while ((ch = string[index++]) == ' ' || ch == '\t') ;
      }
    }
    return (ch);
}

/*****************************************************************************
  stoia : "string to short integer array"
  Todd J. Peltzer
  16 April 1989
  This function takes a character string and converts it to
  an array of short integers.
*****************************************************************************/
stoia(string,array,nbrptr,maxnum)
char string[];
short array[];
int *nbrptr;
int maxnum;
{
  char ch;
  int index=0;
  *nbrptr=0;

  /* strip leading blanks and tabs */
  while ( (ch=string[index++])==' ' || ch == '\t') ;

  while ( (ch >= '0' && ch <= '9') && *nbrptr<maxnum)
  {
    array[*nbrptr] = 0;
    while (ch >= '0' && ch <= '9')
    {
      array[*nbrptr] = 10*(array[*nbrptr]) + ch - '0';
      ch = string[index++];
    }
    (*nbrptr)++;
    while (ch==' ' || ch == '\t')
      ch = string[index++];
  }
  return ch;
}

/*****************************************************************************
   Norbert Doerry
   11 July 1988

   This converts all the upper case characters in a string to lower case
*****************************************************************************/
suctolc(instring,outstring)
char instring[],outstring[];
{
```

```
  int i;

  for (i = 0; instring[i] != NULL ; i++)
    {
      if (instring[i] >= 'A' && instring[i] <= 'Z')
        outstring[i] = instring[i] - 'A' + 'a';
      else
        outstring[i] = instring[i];
    }
  outstring[i] = NULL;
}

/*************************************************************************
   NHD

   This converts all the lower case characters in a string to upper case
*************************************************************************/
slctouc(instring,outstring)
char instring[],outstring[];

{
  int i;

  for (i = 0; instring[i] != NULL ; i++)
    {
      if (instring[i] >= 'a' && instring[i] <= 'z')
        outstring[i] = instring[i] - 'a' + 'A';
      else
        outstring[i] = instring[i];
    }
  outstring[i] = NULL;
}


/*************************************************************************
  This function strips a string of leading and trailing spaces and tabs.
*************************************************************************/
strstrip(s)
char *s;
{
  int i,j;

  /* find first none space or tab */

  for (i = 0 ; s[i] == ' ' || s[i] == '\t' ; i++);

  /* copy string */

  for (j = 0 ; s[i] != NULL ; s[j++] = s[i++]);

  s[j] = NULL;

  /* delete trailing spaces and tabs and Cr*/
```

```
  for (j = strlen(s) - 1 ; s[j] == ' ' || s[j] == '\t' ||
       s[j] == '\n' ; s[j--] = NULL);
}

/*************************************************************************
    This function pads the end of a string with a specified
  number of spaces.
*************************************************************************/
pad(str,n)
char *str;              /* string to pad with spaces */
int n;                  /* number of spaces to pad */
{
  int i=0,j;

  while ( str[++i]!=NULL);
  for (j=0; j<n; j++)
    str[i+j]=' ';
  str[i+n]=NULL;
}

/*************************************************************************
    This function strips the carriage return from a string.
*************************************************************************/
stripcr(s)
char *s;
{
  int j;

  /* delete trailing spaces, tabs, and carriage returns */

  for (j = strlen(s) - 1 ; s[j] == ' ' || s[j] == '\t' ||
       s[j] == '\n' ; s[j--] = NULL);
}

/*************************************************************************
    This function emulates part of the Turbo C bioskey() function.
*************************************************************************/
bioskey(c)
int c;
{
  switch(c) {
        case 0: return get_key();
        case1: return kbhit();
  }
}

/*************************************************************************
    This function reads the 16-bit scan code of a key.
*************************************************************************/
get_key()
{
  union REGS r;

  r.h.ah = 0;
```

```
  return int86(0x16, &r, &r);
}

/*************************************************************************
    This function returns the position code of arrow and function keys.
*************************************************************************/
get_special()
{
  union inkey {
    char ch[2];
    int i;
  } c;

  while (!bioskey(1));      /* wait for keystroke */
  c.i = bioskey(0);         /* read the key */

  return c.ch[1];
}

/*************************************************************************
  This function tests if a character is in a given string; returns the
  position of the character in the string if found, otherwise returns 0.
*************************************************************************/
is_in(s,c)
char *s, c;
{
  register int i;

  for (i=0; *s; i++) if (*s++ == c) return i+1;
  return 0;
}

/*************************************************************************
  This function beeps the speaker using the specified frequency.
*************************************************************************/
void beep1()
{
   int freq=200;
   unsigned i;
   union {
      long divisor;
      unsigned char c[2];
   } count;

   unsigned char p;
   count.divisor = 1193280 / freq;  /* compute the proper count */
   outp(67, 182);  /* tell 8253 that a count is coming */
   outp(66, count.c[0]);  /* send low-order byte */
   outp(66, count.c[1]);  /* send high-order byte */
   p = inp(97);  /* get existing bit pattern */
   outp(97, p | 3);  /* turn on bits 0 and 1 */

   for(i=0; i<32000; ++i);  /* delay loop */
   outp(97, p); /* restore original bits to turn off speaker */
```

```
}

/*************************************************************************
  This function beeps the speaker using the specified frequency.
*************************************************************************/
void beep2()
{
   int freq=400;
   unsigned i;
   union {
      long divisor;
      unsigned char c[2];
   } count;

   unsigned char p;
   count.divisor = 1193280 / freq;  /* compute the proper count */
   outp(67, 182);  /* tell 8253 that a count is coming */
   outp(66, count.c[0]);  /* send low-order byte */
   outp(66, count.c[1]);  /* send high-order byte */
   p = inp(97);  /* get existing bit pattern */
   outp(97, p | 3);  /* turn on bits 0 and 1 */

   for(i=0; i<32000; ++i);  /* delay loop */
   outp(97, p); /* restore original bits to turn off speaker */
}

/*************************************************************************
  video.c
  This file contains a number of functions for controlling screen
  display directly using video RAM.

  Last update: 13 April 1989
-------------------------------------------------------------------------
  Functions:
     set_video()       :  set video memory location
     video_mode()      :  return the current video mode
     save_screen()     :  save a portion of the screen
     restore_screen()  :  restore the same portion of the screen
     goto_xy()         :  send the cursor to row,col
     cls()             :  clear the screen
     clear()           :  clear a portion of the screen
     write_string()    :  display a string with specified attirbute
     write_char()      :  write character with specified attribute
     cursor_off()      :  turn blinking cursor off
     cursor_on()       :  turn blinking cursor on
     display_error()   :  display message in popup window
     pause()           :  pauses until the user presses the INSERT key
     screen_getstrg()  :  takes keystrokes and displays them on the screen
     draw_window()     :  draws a window on the screen
     vid_box()         :  create normal video "boxes"
*************************************************************************/

/*************************************************************************
     This function sets the video memory location.
```

```
*****************************************************************************/
void set_video()
{
  int vmode;

  vmode = video_mode();
  if( (vmode!=2) && (vmode!=3) && (vmode!=7) ) {
    printf("video must be in 80 column text mode");
    exit(1);
  }
  /* set proper address of video RAM */
  if(vmode==7) vid_mem = (char far *) 0xB0000000;
  else vid_mem = (char far *) 0xB8000000;
}

/*****************************************************************************
     This function returns the current video mode.
*****************************************************************************/
video_mode()
{
  union REGS r;

  r.h.ah = 15;          /* get video mode */
  return int86(0x10, &r, &r) & 255;
}

/*****************************************************************************
     This function saves a portion of the screen to memory.
*****************************************************************************/
void save_screen(row1, col1, row2, col2, pt_buffer)
int row1, col1, row2, col2;
char *pt_buffer;
{
    int i, j;
    char far *pt_video;

    for (i=row1; i<=row2; ++i) {
        pt_video = vid_mem + 160*i + 2*col1;
        for (j=col1; j<=col2; ++j) {
            *pt_buffer++ = *pt_video++;         /* Save char. */
            *pt_buffer++ = *pt_video++;         /* Save attribute. */
        }
    }
}

/*****************************************************************************
     This function restores a portion of the screen from memory.
*****************************************************************************/
void restore_screen(row1, col1, row2, col2, pt_buffer)
int row1, col1, row2, col2;
char *pt_buffer;
{
    int i, j;
    char far *pt_video;
```

```
    for (i=row1; i<=row2; ++i) {
        pt_video = vid_mem + 160*i + 2*col1;
        for (j=col1; j<=col2; ++j) {
            *pt_video++ = *pt_buffer++;
            *pt_video++ = *pt_buffer++;
        }
    }
}

/*****************************************************************************
     This function clears the entire screen.
*****************************************************************************/
void cls()
{
  union REGS r;

  r.h.ah=6;             /* screen scroll code */
  r.h.al=0;             /* clear screen code */
  r.h.ch=0;             /* start row */
  r.h.cl=0;             /* start column */
  r.h.dh=24;            /* end row */
  r.h.dl=79;            /* end column */
  r.h.bh=7;             /* blank line is blank */
  int86(0x10, &r, &r);
}

/*****************************************************************************
     This function clears a portion of the screen, using either normal or
  reverse video.
*****************************************************************************/
void clear(row1,col1,row2,col2,attrib)
unsigned row1,row2,col1,col2;
int attrib;
{
  char far *pt_video;
  unsigned row,col;

  for (row=row1; row<=row2; ++row)
  {
    pt_video = vid_mem + 160*row + 2*col1;
    for (col=col1; col<=col2; ++col)
    {
      *pt_video++ = ' ';
      *pt_video++ = attrib;
    }
  }
}

/*****************************************************************************
     This function sends the cursor to a specified location on the screen.
*****************************************************************************/
void goto_xy(row, col)
int row, col;
```

```
{
  union REGS r;

  r.h.ah=2;      /* cursor addressing function */
  r.h.dl=col;    /* column coordinate */
  r.h.dh=row;    /* row coordinate */
  r.h.bh=0;      /* video page */
  int86(0x10, &r, &r);
}

/*************************************************************************
     This function writes a string at a specified location on the screen,
  using a specified attribute.
*************************************************************************/
void write_string(row, col, string, attrib)
int row, col;
char *string;
int attrib;
{
  register int i;
  char far *v;

  v = vid_mem;
  v += (row*160) + col * 2;      /* compute the address */
  for (i=col; *string; i++) {
        *v++ = *string++;        /* write the character */
        *v++ = attrib;           /* write the attribute */
  }
}

/*************************************************************************
     This function writes a string at a specified location on the screen,
  using a specified attribute.
*************************************************************************/
void write_char(row, col, ch, attrib)
int row, col;
char ch;
int attrib;
{
  register int i;
  char far *v;

  v = vid_mem;
  v += (row*160) + col * 2;          /* compute the address */
  *v++ = ch;                         /* write the character */
  *v = attrib;                       /* write the attribute */
}

/*************************************************************************
     This function turns off the blinking screen cursor.
*************************************************************************/
void cursor_off()
{
  union REGS r;
```

```
  r.h.ah=1;      /* cursor size code */
  r.h.ch=0x20;   /* set bit 5 on; turn off cursor */

  int86(0x10, &r, &r);
}

/*****************************************************************************
     This function turns on the blinking screen cursor.
*****************************************************************************/
void cursor_on()
{
  union REGS r;

  r.h.ah=1;      /* cursor size code */
  r.h.ch=6;      /* set start scan line */
  r.h.cl=7;      /* set end scan line */

  int86(0x10, &r, &r);
}

/*****************************************************************************
 Function: display_error()
 Author: Todd J. Peltzer
 Last update: 24 April 1989

    This function creates a popup window and displays up to two lines of
 text, with either "ERROR" or "WARNING" as header, or with a blank header.
 Maximum length of text per line is 38 characters.
*****************************************************************************/
static char footer[] =
{" Press any key to continue "};

void draw_window();

void display_error(type,text1,text2)
int type;
char *text1, *text2;
{
  char *buffer;    /* video buffer     */
  int row1=10;     /* start row        */
  int row2=16;     /* end row          */
  int col1=19;     /* start col        */
  int col2=61;     /* end col          */
  char header[8]; /* header text      */
  int start;

  union inkey {
    char ch[2];
    int i;
  } c;

  cursor_off();
```

```
if (type==ERROR)
(
  sprintf(header,"ERROR");
  beep1();
}
else if (type==WARN)
(
  sprintf(header,"WARNING");
  beep2();
}
else ;         /* type==BLANK */

/* allocate enough memory for video buffer */
buffer = (char *) malloc (2 * (row2-row1+1) * (col2-col1+1) );

/* save portion of video screen */
save_screen(row1,col1,row2,col2,buffer);

/* create window background and border */
draw_window(row1,col1,row2,col2,DOUBLE,REV_VID);

if (type==ERROR || type==WARN)
{
  /* write header */
  start=col1+(col2-col1-strlen(header))/2;
  write_string(11,start,header,BLINK_REV_VID);
)

/* write footer */
start=col1+(col2-col1-strlen(footer))/2;
write_string(row2,start,footer,REV_VID);

/* write first line of error message */
start=col1+(col2-col1-strlen(text1))/2;
write_string(13,start,text1,REV_VID);

/* write second line of error message */
start=col1+(col2-col1-strlen(text2))/2;
write_string(14,start,text2,REV_VID);

/* wait for keystroke, then restore screen and exit */
while (1)
(
  c.i = bioskey(0);        /* read the key */

  if (c.ch[0])             /* key is a normal key */
  {
    break;
  )
  else                     /* key is a special key */
  (
    break;
  )
)
```

```
  restore_screen(row1,col1,row2,col2,buffer);
  free(buffer);
  cursor_on();
)

/*************************************************************************
     This function halts program execution until the user presses the
  INSERT key.
*************************************************************************/
void pause(startrow, startcol, endrow, endcol, msg, attrib)
int startrow;
int startcol;
int endrow;
int endcol;
char *msg;
int attrib;
{
  int start, key, i;

  /* write quit message */
  start=(endcol-startcol-strlen(msg))/2 + startcol;
  write_string(endrow,start,msg,attrib);

  /* get user's response */
  while (1)
  {
    key=get_special();

    if (key==INSERT)
    {
      /* erase message */
      for (i=0; i<strlen(msg); i++)
      {
        write_char(endrow,start+i,205,attrib);
      }
      break;
    )
    else if (key==ALT_Q)      /* exits program; returns to DOS */
    {
      cursor_on();
      cls();
      exit(0);
    }
    else
    {
      continue;
    )
  }
)

/*************************************************************************
     This function takes input keystrokes and displays them on the screen.
  Uses full 16 bit scan code.
```

```
*****************************************************************************/
typedef struct
{
  int startcol;
  int endcol;
} data_box;

void screen_getstrg(i,row,col,string,attrib,input)
int i;           /* data index        */
int row, col;    /* starting row, col */
char *string;    /* ptr to string     */
int attrib;      /* norm/rev video    */
data_box *input; /* data structure    */
{
  union inkey {
    char ch[2];
    int i;
  } c;

  int j=0;       /* array index */
  int k;

  while (1)
  {
    c.i = bioskey(0);       /* read the key */

    if (c.ch[0])
    {
      switch (c.ch[0])
      {
        case ESC  :            /* the ESCAPE key is pressed */
          j=0;                 /* reset array index */
          string[j]=NULL;      /* NULL string */
          for (k=input[i].startcol; k<input[i].endcol; k++)
            write_char(row,k,' ',attrib);
          col=input[i].startcol;
          break;
        case '\r' :            /* the ENTER key is pressed */
          string[j]=NULL;      /* NULL terminate string    */
          return;
        case BKSP :            /* back space */
          if (col == input[i].startcol) break;  /* prevent backspacing past start */
          else
          {
            col--;
            j--;
            write_char(row,col,' ',attrib);
            c.ch[0] = ' ';
            string[j]=c.ch[0];
            break;
          }
        case ' ' :
          if (i>0) break;   /* spaces not allowed in numeric input */
          else if (col == input[i].endcol-1)
```

```
          {
            break;  /* no typing beyond end of box */
          }
          else
          {
            write_char(row,col,c.ch[0],attrib);
            string[j]=c.ch[0];
            col++;
            j++;
            break;
          }
        default :
          if (col == input[i].endcol-1)
          {
            break;  /* no typing beyond end of box */
          }
          else
          {
            write_char(row,col,c.ch[0],attrib);
            string[j]=c.ch[0];
            col++;
            j++;
          }
      }
      goto_xy(row,col);
    }
    else
    {
      switch(c.ch[1])
      {
        case ALT_Q :
          cursor_on();
          cls();
          exit(0);
        default :
          break;
      }
    }
  }
}

/*************************************************************************
     This function draws a window on the screen with a specified video
  attribute (reverse or normal), and a specified border (single or double).
*************************************************************************/
void draw_window(startrow,startcol,endrow,endcol,border,attrib)
int startrow;   /* starting row, screen coordinates */
int startcol;   /* starting col, screen coordinates */
int endrow;     /* ending row, screen coordinates   */
int endcol;     /* ending col, screen coordinates   */
int border;     /* if 1, single border; if 2, double border */
int attrib;     /* normal or reverse video */
{
  int row,col;
```

```
  int upr_lft,upr_rt,lwr_lft,lwr_rt,top,bot,side;

  if (border==SINGLE)
  {
    upr_lft=218;
    upr_rt=191;
    lwr_lft=192;
    lwr_rt=217;
    top=bot=196;
    side=179;
  }
  if (border==DOUBLE)
  {
    upr_lft=201;
    upr_rt=187;
    lwr_lft=200;
    lwr_rt=188;
    top=bot=205;
    side=186;
  }

  /* create background */
  for (row=startrow; row<=endrow; row++)
    for (col=startcol; col<=endcol; col++)
      write_char(row,col,' ',attrib);

  /* if no border, exit */
  if (border==NONE)
  {
    return;
  }

  /* draw border */
  write_char(startrow,startcol,upr_lft,attrib);   /* draw corners */
  write_char(startrow,endcol,upr_rt,attrib);
  write_char(endrow,startcol,lwr_lft,attrib);
  write_char(endrow,endcol,lwr_rt,attrib);

  for(col=startcol+1; col<endcol; col++)        /* draw top border */
    write_char(startrow,col,top,attrib);
  for(col=startcol+1; col<endcol; col++)        /* draw bottom border */
    write_char(endrow,col,bot,attrib);
  for(row=startrow+1; row<endrow; row++)        /* draw left border */
    write_char(row,startcol,side,attrib);
  for(row=startrow+1; row<endrow; row++)        /* draw right border */
    write_char(row,endcol,side,attrib);
}

/*****************************************************************************
     This function draws text header on the screen with a specified video
  attribute (reverse or normal), and a single border.
*****************************************************************************/
void write_header(startrow,startcol,endcol,header,attrib)
int startrow;
```

```
int startcol;
int endcol;
char *header;
int attrib;
{
  int start;

  start = (endcol-startcol-strlen(header))/2 + startcol;
  draw_window(startrow+1,start-1,startrow+3,start+strlen(header),
               SINGLE,attrib);
  write_string(startrow+2,start,header,attrib);
}
/*************************************************************************
     This function is used to create normal video "boxes" on an otherwise
  reverse video background to highlight data entry locations.
*************************************************************************/
void vid_box(startrow,startcol,length)
int startrow;
int startcol;
int length;
{
  int i;

  /* create normal video boxes for data entry */
  for (i=0; i<length; i++)
    write_char(startrow,startcol+i,' ',NORM_VID);
}


/*************************************************************************
  popup.c
  Author: Todd J. Peltzer
  Last update: 29 March 1989

-------------------------------------------------------------------------
  Functions:
    popup()
    display_menu()
    get_resp()
*************************************************************************/

void display_menu(), draw_window();

/*************************************************************************
   This function displays a pop-up menu and returns the user's selection:
     -- returns -2 if menu cannot be constructed
     -- returns -1 if user hits escape key
     -- otherwise the item number is returned starting
          with 0 as the first (top most) entry
*************************************************************************/
int popup(menu,keys,count,startrow,startcol,border,attrib,start)
char *menu[];    /* menu text */
char *keys;      /* hot keys  */
int count;       /* number of menu items */
```

```
int startrow;   /* starting row, screen coordinates */
int startcol;   /* starting col, screen coordinates */
int border;     /* if 0, no border; if 1, single border; if 2, double border */
int attrib;     /* normal or reverse video */
int start;      /* starting item to highlight */
{
  register int i, len;
  int endrow;     /* ending row, screen coordinates   */
  int endcol;     /* ending col, screen coordinates   */
  int choice, vmode;
  unsigned char *p;  /* buffer for screen data */

  if( (startrow>24) || (startrow<0) || (startcol>79) || (startcol<0) )
  {
        printf("range error");
        return -2;
  )

  /* compute dimensions */
  len = 0;
  for(i=0; i<count; i++)
    if(strlen(menu[i]) > len) len = strlen(menu[i]);
  endcol = len + 4 + startcol;
  endrow = count + 1 + startrow;
  if ((endrow+1>24) || (endcol+1>79))
  {
    printf("menu won't fit");
    return -2;
  }


  /* allocate enough memory for menu screen buffer*/
  p = (unsigned char *) malloc(2*(endrow-startrow+1)*(endcol-startcol+1));
  if(!p) exit(1); /* install error handler here */

  /* save the current screen data */
  save_screen(startrow,startcol,endrow,endcol,p);

  /* draw border and background */
  draw_window(startrow,startcol,endrow,endcol,border,attrib);

  /* display the menu */
  display_menu(menu,startrow+1,startcol+2,count,attrib);

  /* get the user's response */
  choice = get_resp(startrow+1,startcol+1,count,menu,keys,attrib,start);

  /* restore the original screen*/
  restore_screen(startrow,startcol,endrow,endcol,p);
  free(p);
  return choice;
}

/*************************************************************************
```

```
     This function displays the menu in its proper location
/*************************************************************************/
void display_menu(menu,row,col,count,attrib)
char *menu[];
int row;
int col;
int count;
int attrib;
{
  register int i;

  for(i=0; i<count; i++, row++) {
        write_string(row,col, menu[i], attrib);
  )
)

/*************************************************************************
     This function gets the user's selection.
/*************************************************************************/
get_resp(row,col,count,menu,keys,attrib,start)
int row, col, count;
char *menu[];
char *keys;
int attrib;
int start;
{
  union inkey {
        char ch[2];
        int i;
  ) c;
  int arrow_choice=start, key_choice;
  int highlight;

  col++;

  /* highlight the first selection */
  if (attrib==NORM_VID)
  {
    highlight = REV_VID;
  }
  else if (attrib==REV_VID);
  {
    highlight = NORM_VID;
  )
  goto_xy(row+start, col);
  write_string(row+start, col, menu[0+start], highlight);

  for(;;)
  {
    while(!bioskey(1)) ; /* wait for key stroke */
    c.i = bioskey(0);    /* read the key */

    /* reset the selection to original video attribute */
    goto_xy(row+arrow_choice, col);
```

```
    write_string(row+arrow_choice,col,menu[arrow_choice],attrib);

    if(c.ch[0])     /* is normal key */
    {
      /* see if it is a hot key */
      key_choice = is_in(keys, tolower(c.ch[0]));
      if(key_choice) return key_choice-1;

      /* check for ENTER or space bar */
      switch(c.ch[0])
      {
        case '\r': return arrow_choice;
        case ' ' : arrow_choice++;
            break;
        case ESC : return -1; /* cancel */
      }
    }
    else        /* is special key */
    {
      switch(c.ch[1])
      {
        case UP_ARROW:
          arrow_choice--; /* up arrow */
          break;
        case DOWN_ARROW:
          arrow_choice++; /* down arrow */
          break;
        case HOME:
          arrow_choice=0;
          break;
        case END:
          arrow_choice=count-1;
          break;
      }
    }
    if(arrow_choice==count) arrow_choice=0;
    if(arrow_choice<0) arrow_choice=count-1;

    /* highlight the next selection */
    goto_xy(row+arrow_choice,col);
    write_string(row+arrow_choice,col,menu[arrow_choice],highlight);
  }
}
```

214-889

Thesis
P3285
c.1

Peltzer
Computer-aided decision making for ocean towing.

Thesis
P3285
c.1

Peltzer
Computer-aided decision making for ocean towing.

AMERICAN BINDERIES

thesP3285
Computer-aided decision making for ocean
3 2768 000 82223 3
DUDLEY KNOX LIBRARY